REJ11B0006-0102

Everywhere you imagine. RENESAS

M16C/6S DATA LINK LAYER LIBRARY D2DL USER'S MANUAL

RENESAS SINGLE-CHIP MICROCOMPUTER
M16C Family /M16C/6S Group

Rev 1.02
Jun 30, 2006

Renesas Technology
www.renesas.com

[Contents]

1. Introduction

This document is the User's Manual of communication interface library (hereafter called the D2DL) for Renesas Technology's PLC (Power Line Communication) microcomputer M16C/6S. The D2DL achieves the function of the Physical Layer (the first layer) and the Data Link Layer (the second layer) in the OSI network reference model. This document provides the necessary information which helps you understand and use the D2DL. From section 1 to 3, it describes the D2DL overview and the basic information. After section 4, it describes the necessary information when you develop some programs with the D2DL.

The M16C/6S integrates IT800 PLC modem technology developed by Yitran Communications Ltd., which enables extremely robust PLC communication. The D2DL is a communication library which optimized for IT800.
The D2DL provides some API functions for the interface with the upper layer. You can make some PLC communication programs easily to use these API functions.

The D2DL runs on the real-time OS (hereafter called the RTOS), and we provide two types of edition, "Single Task Edition" and "Multi Task Edition" for each user's usage.
You can use the Single Task Edition for your system development that needs only one task (i.e. it doesn't need multiple tasks).
If you develop a system that needs multiple tasks, you have to use the Multi Task Edition.
This document gives a description both of "Single Task Edition" and "Multi Task Edition".

The following table shows some documents about the D2DL and their descriptions.

Document Name	Description
D2DL USER'S MANUAL	This document has information which helps you understand and use the D2DL. **We recommend you read this document first.**
D2DL QUICK START GUIDE	It describes the operation to run the example program attached to the D2DL.
D2DL API SPECIFICATION	It gives a detailed description of the API functions for D2DL.
D2DL EEPROM I/F SPECIFICATION	The D2DL provides the basic function, which accesses the microcomputer hardware and controls the EEPROM, for EEPROM I/F, therefore you can use any EEPROM. The document describes the specification about the functions and the example program attached to the D2DL.
D2DL SPECIFICATION	It describes the communication feature of the D2DL.

2. D2DL Overview

2.1 D2DL feature

The D2DL features are described as follows.

- IT800 PLC modem technology developed by Yitran Communications Ltd. enables extremely robust PLC communication. The D2DL is a communication library which optimized for IT800. If you use the D2DL, you can get the communication which uses the IT800 feature and the unique identity at a maximum and also keep the coexistence with the other IT800 module.
- It is all achieved by the interfaces with the upper layer (user application), such as the transmit/receive packet passing and the parameter setting, by the API functions (and callback functions). You can develop the communication application program easily by these API functions.
- The D2DL runs on RTOS. We provide two types of edition, "Single Task Edition" and "Multi Task Edition". For more details, please refer to the next section.
- You can achieve one-chip solution using the available resource area of microcomputer, which the D2DL does not use, and create the system at low cost.
- You can create the debug environment at once and debug efficiently using the project format of the RENESAS HEW (High-performance Embedded Workshop)

2.2 Compartmentalization between the Single Task Edition and the Multi Task Edition

The D2DL consists of the following three tasks. Therefore the software is the Multi-Task based program.

- Work Task
  It is the main process for PLC communication. This task is invoked periodically.
- Reception Task
  It processes the received PLC data.
- Transmission Task
  It processes the transmitting PLC data from the user application.

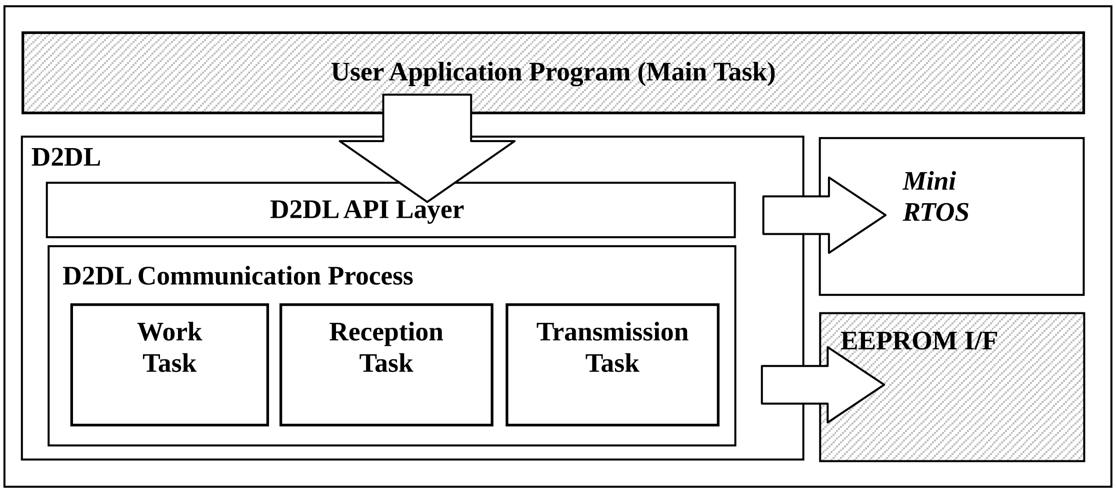
To operate the three tasks mentioned above independently, the RTOS is required. If you use the RTOS, you can get not only these communication processes but also your application program as Multi-Task based program. This means you can divide your application program into some tasks (if you want).
However some users don't want to divide their application program into some tasks, because they prefer Single-Task based programming. **In the Single-Task based programming, the users only create one main loop, some sub-functions which is called from main loop, and some interrupt service routines.** For such users, providing RTOS is not welcome, because it causes additional costs and time for the RTOS which is needed only for the D2DL.
This "D2DL Single Task Edition" is provided to solve this problem. If you use "D2DL Single Task Edition", you do not need to get the RTOS because the edition includes the dedicated mini RTOS (hereafter called MiniRTOS), and you can program your application as Single-Task based program.
To get a Multi-Task based application, you should use the "D2DL Multi Task Edition". In this case, you have to prepare our recommended RTOS (MR30 developed by RENESAS) because "D2DL Multi Task Edition" does not include the RTOS.

2.3 The Internal Block Diagram

This section describes the internal block diagram of the D2DL Single Task Edition and the D2DL Multi Task Edition. The meshed blocks indicate the block the user has to make or prepare.

2.3.1 The Internal Block Diagram of the D2DL Single Task Edition

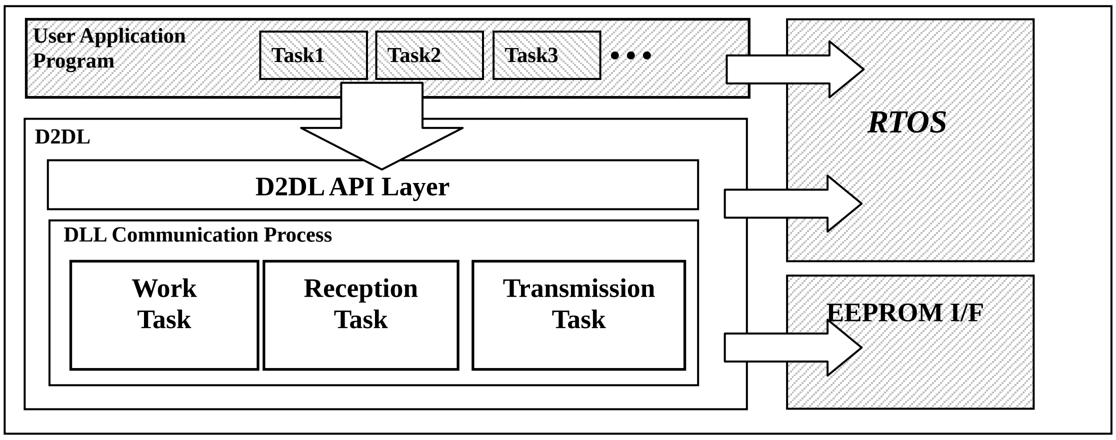
The Internal Block Diagram of the D2DL Single Task Edition is as follows.



The "User Application Program" block is a task in the D2DL Single Task Edition. But user does not need to care about Multi-Task based programming with the RTOS. Therefore, the user can program the User Application which includes **a main loop, some interrupt service routines and some sub-functions which are called by them**.
The user cannot use the features of MiniRTOS from "User Application Program".

2.3.2 The Internal Block Diagram of the D2DL Multi Task Edition

The Internal Block Diagram of the D2DL Multi Task Edition is as follows.



The user can develop the "User Application Program" by the Multi-Task based programming technique with RTOS feature.

[Note]
Please refer to the section 5.3 for the EEPROM interface.

2.4 The resources for the D2DL

The following table shows the microcomputer resources for the D2DL.

	D2DL Single Task Edition	D2DL Multi Task Edition
ROM	41Kbytes (including RTOS)	41Kbytes (including RTOS)
RAM	The user application can set the RAM usage (i.e. the size of the buffer for the transmission and reception packets) The sample program are using 6.5KB RAM.	The user application can set the RAM usage (i.e. the size of the buffer for the transmission and reception packets) The sample program are using 6.5KB RAM.
Timer	TA0,TA1,TA2	TA0,TA1,TA2 (The RTOS uses the TA2)
Serial I/O	SI/O4 (Using for the internal interfaces between the D2DL and the IT800PHY)	SI/O4 (Using for the internal interfaces between the D2DL and the IT800PHY)
DMA	DMA1 (Using for the internal interfaces between the D2DL and the IT800PHY)	DMA1 (Using for the internal interfaces between the D2DL and the IT800PHY)
I/O Port	P4, P5 (Using for the internal interfaces between the D2DL and the IT800PHY)	P4, P5 (Using for the internal interfaces between the D2DL and the IT800PHY)
Interrupt	Refer the next table.	Refer the next table.

The following table shows the details of the interrupts for the D2DL.

Causes of interrupts	Function name	Usage	Priority
SI/O4	D2dll_PhyRx_PhyUsartISR	The communication between the D2DL and the IT800PHY	4
INT0	D2dll_Phy_ISR	The event from the IT800PHY	6
TA0	D2dll_Timer_ISR	The timer for the PHY layer	5
TA1	D2dll_Timer1_ISR	The timer for the DLL layer	3

3. Description of the D2DL

3.1 API

The API functions are defined in the D2DL as the interface with the upper layer (the user application). The next table shows the main features of the API functions and their descriptions.

Feature	Description
Initialization	It initializes and starts the D2DL.
Register an application	It registers an application. On multiple application system, the D2DL can identify each upper application and communicate with each application.
Transmission	It sends a packet. When you check the transmission result, you can select between two methods. One is the method using the callback function. The other is the method that the application waits for the determined transmission result in the transmission function and refers the return value.
Reception	It receives a packet. When you get the received packet, you can select between two methods. The first method is using the callback function and the second one initates the upper layer to call the reception function.
EEPROM access	It accesses the user area of the EEPROM.
Parameter control	It accesses the parameter that the D2DL use.

For details, please refer to the section 4 and the”D2DL API SPECIFICATION”.

3.2 PLC communication feature

This section describes the PLC communication feature of the D2DL.

3.2.1 Packet delivery service

This section describes the D2DL feature about the packet delivery.

3.2.1.1 ACK/UNACK service

The ACK service allows for checking the accession of the transmitted packet at the target node (the destination node). The target node which receives the ACK service packet sends an ACK packet which shows the successful reception. The source node recognizes the transmission success by the reception of the ACK packet. If the source node cannot receive the ACK packet, it retransmits the packet predefined times. If the retransmission failed in the predefined times, an error is returned to the user application.
The UNACK service does not check the ACK packet. It can transmit a packet predefined times to improve the reliability. In this case, the D2DL transfers a received packet only once to the user application regardless of whether the destination node receives the same packet a number of times.
The retransmission times of each service can be set by the API function “D2DLL_SendOption”.

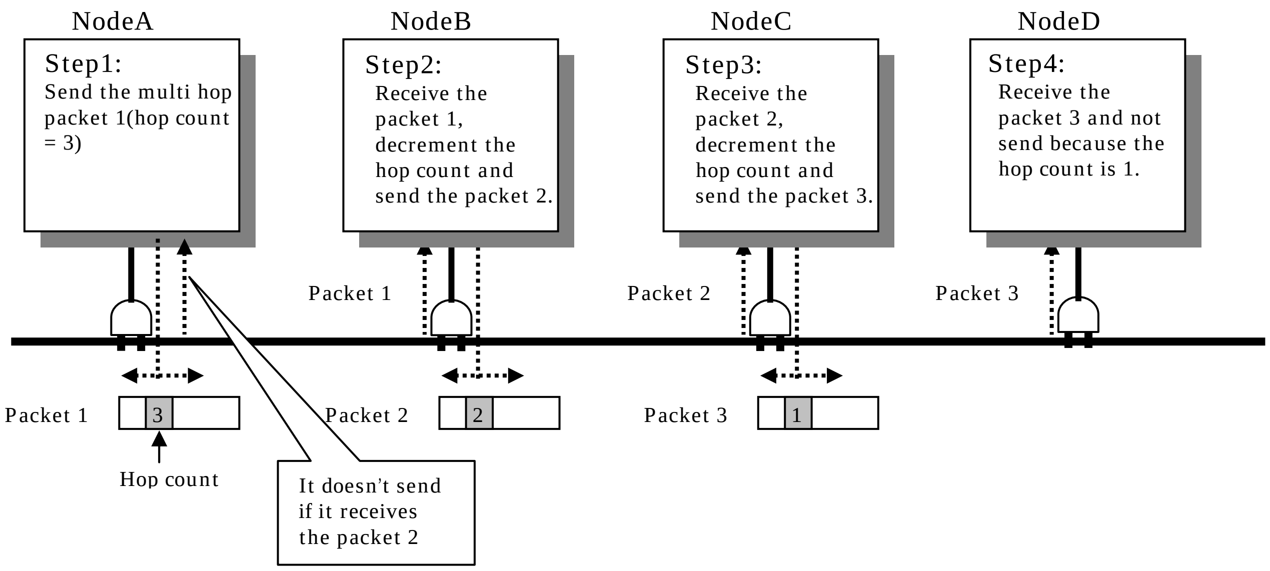
3.2.1.2 Broadcast service

The single network broadcast service can transmit a packet to all nodes in the same logical network (which node has the same network ID).
The CNC (Control Network Channel) service can transmit a packet to all nodes in the physical network regardless of the network ID (even if the node does not have any network ID). It is useful when it assigns an address to a new node. The CNC packet has a DSN (Device Serial Number) of the source node. When the D2DL on the received side node receives a CNC packet, it passes the received information which includes the source DSN to the user application.

3.2.1.3 Multi hop feature

The D2DL has a **multi hop feature** which repeats (hops) the received broadcast packets. When the distance between nodes is too far (or the communication environment is bad) and the signal does not reach directly, this feature is effective. The router nodes relay the packet, so the packet can be reached to the destination node.
The number of times of relay (hop count) is set by the user application of the source node, and included in the packet. The relay node decrements the hop count, set it in the packet and transmit it. If the hop count is 1, the

received node does not relay it.
When a node which relays a packet does receive the same packet again (i.e. it receives the packet which is relayed by the other node,) it doesn't relay it again.
The D2DL relays the packet internally, so the user application does not need to care about it.


NodeA
NodeB
NodeC
NodeD
Step1: Send the multi hop packet 1(hop count = 3)
Step2: Receive the packet 1, decrement the hop count and send the packet 2.
Step3: Receive the packet 2, decrement the hop count and send the packet 3.
Step4: Receive the packet 3 and not send because the hop count is 1.
Packet 1
Packet 2
Packet 3
Packet 1
3
Packet 2
2
Packet 3
1
Hop count
It doesn't send if it receives the packet 2

3.2.1.4 Fragmentation and Reassembly

The D2DL implements a mechanism of **fragmentation** and **reassembly**. The maximum D2DL payload is 110 bytes. When the user application requests the transmission of a packet which size is above 110 bytes, the D2DL fragments it and sends the fragmented data. The D2DL in the receiving node reassembles the received fragmented packets and passes it on to the user application.
The D2DL supports 16 fragments of 110 bytes for a message length of 1760 bytes as a maximum of a long packet.

3.2.1.5 Transmission rate

The IT800PHY supports three transmission modes as follows:
(Note: In Europe, it supports two modes.)

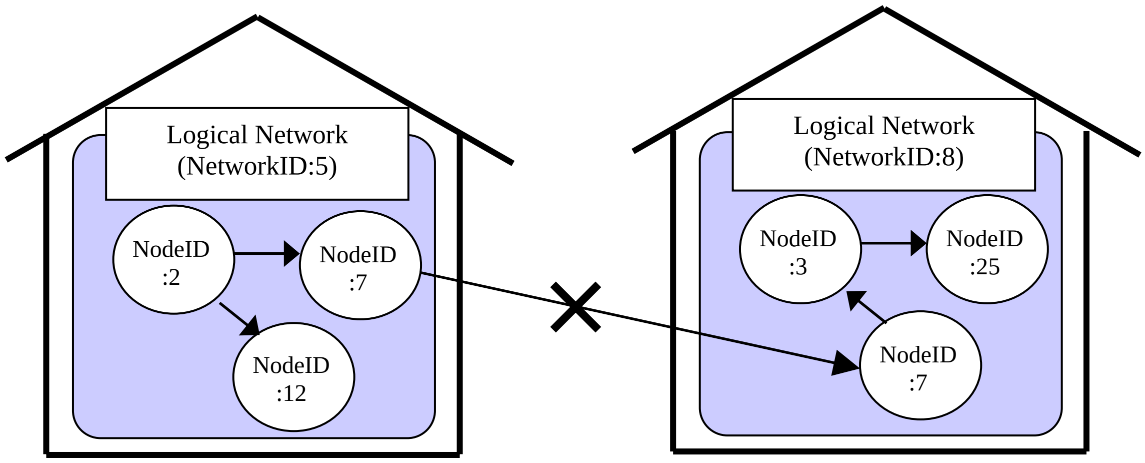
	N. America and Japanese regional settings	Europe regional settings
Standard Mode (SM)	7.5Kbps	-
Robust Mode (RM)	5.0Kbps	2.5Kbps
Extremely Robust Mode (ERM)	1.25Kbps	0.625Kbps

Additionally, the D2DL has the data rate control algorithm which can monitor the communication status of each destination node and select optimal transmission mode automatically. You can select the fixed three mode transmission, mentioned above, or the auto rate control mode transmission.

3.2.2 Addressing

3.2.2.1 Network ID and Node ID

The D2DL uses the network ID (10 bit) and the node ID (11 bit) for the address of each node for general communication. The network ID is assigned to each logical network. It assigns the common network ID to the node in the same logical network. The node ID is assigned to each node and it should be unique in the logical network.
For example, if each house has the different network ID, interference between the houses can be avoided.



3.2.2.2 Device Serial Number (DSN)

The device serial number (DSN) is used as the address in case of the CNC service. The DSN is a unique 16 bytes value which should be assigned to each node, and is stored in the EEPROM of each node. The user application in the node that receives a CNC packet can recognize the source device by the source DSN that is included in the received packet.
The DSN should be assigned to avoid CNC packet overlappings between each user's products. To achieve this necessity, a customer number is assigned to each user which is included in the DSN, whereby DSN overlappings in the D2DL are avoided. For details, please refer to 5.5.

3.2.2.3 Port ID (Application registration)

Multiple processes/works (applications) can share a D2DL. Therefore you can lay out multiple applications on a chip. You can make the number of application up to 16, and identify each application by the port ID (from 0 to 15). This method is useful when you want to identify the processes/works of the destination node, for example when you want to send the packet to the specific process in the specific node.
In the case of using the packet receive callback function (for details, please refer to the section 6 and 7), the D2DL sorts the received packet to the destination application.

3.2.2.4 Protocol number

The D2DL keeps the coexistence of any products which implement different upper protocol by using the protocol number. The nodes which use the same protocol number can communicate with each other, but the nodes which use different protocol number cannot. For example, when there are two nodes which have the same address (network ID and node ID) on the line, they can keep the coexistence if each of them uses the different protocol number.
If you want to use this feature, you should set up the protocol number according to your protocol. For details, please refer to section 5.4.

3.2.3 Media Access Control

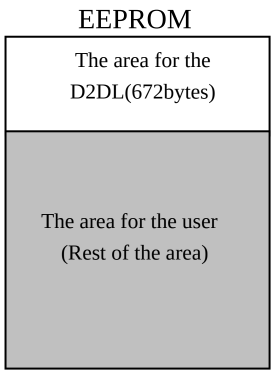
The D2DL supports the channel access control based on the CSMA/CA(Carrier Sense Multiple Access with Collision Avoidance). It implements an adaptive back-off algorithm. The algorithm estimates the number of nodes which contend the channel priority, and optimizes the packet transmission interval.
The transmission priority of the packet can be set up to 4 different priority levels (High/Normal/AboveLow/Low). But the user application can only set 3 priority levels **"High/Normal/Low"** for the transmission packet. The priority "AboveLow" is assigned automatically to the fragmented packets, so this cannot be set by the user application.

3.2.4 Virtual Jamming (Imposter node detection feature)

The D2DL supports a feature to detect the packet from the imposter node attempting to get into the network using the address of a valid node. The D2DL passes the received packet from the imposter node as the **imposter packet** to the user application if you have set the received packet type as the imposter packet by the D2DLL_Start function.

3.3 EEPROM Control feature

The D2DL can store internal parameters to the EEPROM. Therefore, it enables keeping the value of various parameters even if the power is shut down and managing the dedicated serial number per each device, etc. Additionally, the user application can use available areas, which is not used by the D2DL. The D2DL uses the top of 672 bytes of EEPROM area, so the application can use the subsequent area.


EEPROM
The area for the D2DL(672bytes)
The area for the user (Rest of the area)

The D2DL provides the API for the user area access; therefore the user application can access the user area easily.
Please refer to section 5.2 about the data to write to the area for the D2DL.

4. API overview

Developers of application software can develop the program using the D2DL APIs. The application can access to the D2DL via the dedicated API functions. The API functions for the D2DL and the structures used as the arguments for the API functions are described in this section.

4.1 Initialization and termination function API

API name	Parameter		Return value		Description
D2DLL_Init(void *rsc)	*rsc	Resource information in the D2DL	Normal	D2DLL_E_OK	This function initializes the RTOS function for the D2DL with information specified by the argument "rsc". Please set "rsc" to NULL regardless of the single task edition or the multi task edition. This function has to be called once after power is on. Other functions can not be used until this function is called.
			Error	D2DLL_E_PARAM	
				D2DLL_E_TIMING	
				D2DLL_E_SYS	
D2DLL_Start(uint16 netId, uint16 nodeId, sint16 rxPktType, sint16 region, uchar *dsn, sint16 enableRep)	netId	Network ID for my device	Normal	D2DLL_E_OK	This function initializes and starts the D2DL. Each valid value which is set by the argument of this function is saved to EEPROM in this function. This function can be called after the **D2DLL_Init** function and the **D2DLL_RegApp** function, and before the **D2DLL_Online** function.
	nodeId	Node ID for my device	Error	D2DLL_E_PARAM	
	rxPktType	Receive data type		D2DLL_E_TIMING	
	region	Area configuration		D2DLL_E_SYS	
	*dsn	Device Serial Number (16byte value) for my device			
	enableRep	Enable/Disable repeater function			
D2DLL_RegApp(uint16 portId, d2dll_fpOnRxPktCb_t fpOnReceived, d2dll_fpOnTxResCb_t fpOnTransmitted)	portId	Port ID of the application	Normal	D2DLL_E_OK	This function registers the current application. Each application has a different port ID. This function can be called after the **D2DLL_Init** function and has to be called at least once before the **D2DLL_Start** function. After calling the **D2DLL_Start** function, you can call this function at any time.
	fpOnReceived	Pointer to the function which will be called back when the D2DL receive a packet	Error	D2DLL_E_PARAM	
				D2DLL_E_TIMING	
	fpOnTransmitted	Pointer to the function which will be called back when the D2DL has transmitted a packet for this application		D2DLL_E_SYS	
D2DLL_Online(void)	void	-	Normal	D2DLL_E_OK	This function enables the use of the PLC communication. This function can be called after calling the **D2DLL_Start** function or the **D2DLL_Offline** function, and has to be called before starting the PLC communication.
			Error	D2DLL_E_TIMING	
				D2DLL_E_SYS	
D2DLL_Offline(void)	void	-	Normal	D2DLL_E_OK	This function terminates the PLC communication.

			Error	D2DLL_E_TIMING D2DLL_E_SYS	All functions in transmit waiting state and receive waiting state will be terminated compellingly with the retun value "D2DLL_E_SYS". This function can be called when the PLC communication is enabled by the **D2DLL_Online** function. If you want to enable the PLC communication again, the **D2DLL_Online** function must be called.

4.2 Transmission function API

API name	Parameter			Return value		Description
D2DLL_Send(d2dll_sndParam *sndParam)	*sndParam	srcPortId	PortID of source application	Normal Error	D2DLL_E_OK D2DLL_E_PARAM D2DLL_E_TIMING	This function transmits the data. [Notes] In the case of the Multi Task Edition, each task can call this function at the same time. However the function called later is set to WAIT state internally by the semaphore function. The argument "fpOnTransmitted" of the **D2DLL_RegApp** function defines the pointer of the function to get the result of transmission. If it is set to NULL, the result of transmission is passed as a return value of the **D2DLL_Send** function, therefore, the **D2DLL_Send** function does not return until the transmission sequence is completed. If the argument is not set to NULL (i.e. it is set to the pointer to the callback function), the result of transmission is passed by the callback function. In this case, the **D2DLL_Send** function returns soon after issuing the transmission request to lower layer. This function can be called when the PLC communication is enabled by the **D2DLL_Online** function.
		dstNodeId	Destination Node ID		D2DLL_E_SYS	
		dstPortId	Port ID of the destination application		D2DLL_E_NOACK D2DLL_E_BLCKD D2DLL_E_NORESRC	
		*sndData	Transmission data			
		sndDataLen	Length of transmission data			
		sessionTag	Packet ID			
		sndPrty	Transmission priority			
		sndAck	ACK/UNACK setting			
		*s_extnsn	Reserved for future			
D2DLL_Send_MH(d2dll_sndParam *sndParam, uint16 sndMHcnt	*sndParam	---The same as D2DLL_Send	---The same as D2DLL_Send	---The same as D2DLL_Send	---The same as D2DLL_Send	This function enables the multiple hop broadcast packet transmission. The multiple hop feature allows for repeating (hopping) of the received multihop broadcast packet.
	sndMHcnt	Hop counter				

)						[Note] The same as D2DLL_Send
D2DLL_Send_CNC(d2dll_sndParam *sndParam, uint16 sndMHcnt)	*sndParam	---The same as D2DLL_Send	---The same as D2DLL_Send	---The same as D2DLL_Send	---The same as D2DLL_Send	This function enables the CNC (Control Network Channel packet type) packet broadcast transmission. This type of the packet is used to broadcast messages to all the connected nodes regardless of their network ID (if they don’t have their network ID). The CNC service is useful for introducing a new node to a network. The CNC packet contains the DSN of the source node. [Note] The same as D2DLL_Send
	sndMHcnt	Hop counter				
(*d2dll_fpOnTxResCb_t)(sint16 sessionTag, sint16 sndResult)	sessionTag	Packet ID		-	-	This function is a callback function, so this function has to be implemented in a user application. When the D2DL completed the transmission sequence, this function will be called for giving the result of transmission to the user application. If you do not want to use this callback function, set the argument “fpOnTransmitted” of **D2DLL_RegApp** function to NULL and get the result as the return value of packet transmission function. This function is enabled when the PLC communication is enabled by the **D2DLL_Online** function.
	sndResult	Result of Transmission		-	-	

4.3 Reception function API

API name	Parameter			Return value		Description
D2DLL_Recv(d2dll_rcvParam *rcvParam)	*rcvParam	rcvPcktType	Type of the received packet	Normal	Positive value: Size of the receive data 0: There is no data	This function is for data reception. If you get the received data using the callback function, this function is unnecessary for you. The return timing from this function depends on the specified receive time-out. In the case of the Multi Task Edition, multiple tasks can call this function at the same time. However when a task (function) is running to receive, the function called later is set to WAIT state internally by the semaphore function. This function can be called when the PLC
		srcNetId	Source Network ID	Error	D2DLL_E_PARAM	
					D2DLL_E_TIMING	
		srcNodeId	Source Node ID		D2DLL_E_SYS	
		srcPortId	Port ID of the source application		D2DLL_E_TMOUT	
		dstNodeId	Destination Node ID			
		dstPortId	Port ID of the			

			destination application			communication is enabled by the **D2DLL_Online** function.
		*rcvData	Start address of the user receive buffer(for the D2DLL_Recv function)			
			Start address of the D2DL receive buffer (for the receive callback function)			
		rcvDataLen	Size of the user receive buffer(for the D2DLL_Recv function)			
			Size of the receive buffer(for the receive callback function)			
		rcvTimeout	Setting receive time-out(Unit:1msec)			
		rcvSQuality	Signal Quality			
		r_option	Option parameter			
		*r_extnsn	Future reserved			
(*d2dll_fpOnRxPktCb_t) (d2dll_rcvParam *rcvParam)	*rcvParam	---The same as D2DLL_Recv	---The same as D2DLL_Recv	-	-	This function is a callback function, so this function has to be implemented in a user program. When the D2DL has received the packet from the power line, this function will be called for the data reception. If you do not want to use this callback function, set the argument “fpOnReceived” of the **D2DLL_RegApp** function to NULL and get the received data using the **D2DLL_Recv** function. This function is enabled when the PLC communication is enabled by the **D2DLL_Online** function.

4.4 EEPROM access API

API name	Parameter		Return value		Description
D2DLL_SizeEep (uint16 size)	size	Memory size of your EEPROM (byte)	Normal	D2DLL_E_OK	This function passes the size of your EEPROM to the D2DL. Please refer to "D2DL EEPROM I/F SPECIFICATION" for details. This function can be called at any time after power on.
			Error	D2DLL_E_PARAM	
D2DLL_WriteEep(uint16 addr, uint16 size, uchar *buff)	addr	Address to write in EEPROM	Normal	D2DLL_E_OK	This function enables to write the data to the user area in EEPROM. Please refer to "D2DL EEPROM I/F SPECIFICATION" for details. This function can be called when the PLC communication is enabled by the **D2DLL_Online** function.
	size	Size of the data to write	Error	D2DLL_E_PARAM	
	*buff	Pointer to the buffer which stores the data to write		D2DLL_E_TIMING	
				D2DLL_E_SYS	
D2DLL_ReadEep(uint16 addr, uint16 size, uchar *buff)	addr	Address to read in EEPROM	Normal	D2DLL_E_OK	This function enables to read the data from the user area in EEPROM. Please refer to "D2DL EEPROM I/F SPECIFICATION" for details. This function can be called after calling the **D2DLL_SizeEep** function.
	size	Size of the data to read	Error	D2DLL_E_PARAM	
	*buff	Pointer to the buffer which stores the data to read		D2DLL_E_TIMING	
				D2DLL_E_SYS	
D2DLL_SaveParam (void)	void	-	Normal	D2DLL_E_OK	This function copies all D2DL parameters from RAM to EEPROM. The D2DL parameter means the parameters which are set by the following functions. - D2DLL_SetParam - D2DLL_SetAddrs - D2DLL_SendOption This function can be called after calling the **D2DLL_Init** function.
			Error	D2DLL_E_TIMING	
				D2DLL_E_SYS	

4.5 Parameter Control API

API name	Parameter		Return value		Description
D2DLL_SendOption(uint16 ackRetry, uint16 repCnt, sint16 sndRate	ackRetry	Numbers of retransmission in ACK mode	Normal	D2DLL_E_OK	This function sets the transmission parameter. The D2DL has a capability to store the argument values to EEPROM. But this function does not store these values to EEPROM. It changes only
	repCnt	Numbers of retransmission in UNACK mode	Error	D2DLL_E_PARAM	

)	sndRate	Transmit rate (Rates are different for each area)		D2DLL_E_TIMING D2DLL_E_SYS	data in RAM. If you want to store them to EEPROM, you have to call the **D2DLL_SaveParam** function after calling this function. This function can be called when the PLC communication is enabled by the **D2DLL_Online** function.
D2DLL_SetParam(sint16 index, uint16 value)	index value	Index of the entry Entry value	Normal Error	D2DLL_E_OK D2DLL_E_PARAM D2DLL_E_TIMING D2DLL_E_SYS	This function sets one entry in the D2DL parameter table. Please use only our recommend index values. - D2DLL_IDX_PROTOCOL_VER(70) Protocol number - D2DLL_IDX_DLL_MEMORY (71) The size of the sent/received buffer(byte) - D2DLL_IDX_RXRES_MEMORY (77) The size of the special area for the received packet(byte) This function can be called after calling the **D2DLL_Init** function.
D2DLL_GetParam(sint16 index, uint16 *value)	index *value	Index of the entry Entry value	Normal Error	D2DLL_E_OK D2DLL_E_PARAM D2DLL_E_TIMING D2DLL_E_SYS	This function reads one entry in the D2DL parameter table. Please use only our recommend value of the index. This function can be called after calling the **D2DLL_RegApp** function.
D2DLL_GetVer(uint32 *version)	*version	Version of the D2DL	Normal Error	D2DLL_E_OK D2DLL_E_PARAM D2DLL_E_TIMING D2DLL_E_SYS	This function returns the version of the D2DL. This function can be called when the PLC communication is enabled by the **D2DLL_Online** function.
D2DLL_SetAddrs(uint16 netId, uint16 nodeId)	netId nodeId	Network ID for my device Node ID for my device	Normal Error	D2DLL_E_OK D2DLL_E_PARAM D2DLL_E_TIMING D2DLL_E_SYS	This function sets own network ID and own node ID. The address is also able to be set by the **D2DLL_Start** function. The D2DL has a capability to store the network ID and the node ID to EEPROM. But this function does not store these values to EEPROM. It changes only the data in RAM. If you want to store them to EEPROM, you have to call the **D2DLL_SaveParam** function after calling this function.

					This function can be called when the PLC communication is enabled by the **D2DLL_Online** function.
D2DLL_GetAddrs(uint16 *netId, uint16 *nodeId)	netId	Network ID for my device	Normal	D2DLL_E_OK	This function returns own network ID and own node ID. This function can be called after calling the **D2DLL_RegApp** function.
	nodeId	Node ID for my device	Error	D2DLL_E_PARAM	
				D2DLL_E_TIMING	
				D2DLL_E_SYS	

4.6 Error Code

Error codes in the API functions for the D2DL are shown as follows.

Name	Value	Description
D2DLL_E_OK	0	Normal end
D2DLL_E_PARAM	-1	Parametric error
D2DLL_E_TIMING	-2	Invalid call
D2DLL_E_SYS	-3	System error (i.e. internal system error in OS etc.)
D2DLL_E_NOACK	-10	No ACK response
D2DLL_E_BLCKD	-11	Outgoing packet is blocked for congestion
D2DLL_E_NORESRC	-12	No resource (currently, not enough available memory to accept packet)
D2DLL_E_TMOUT	-20	Time-out

5. Development of the User Application (For both edition)

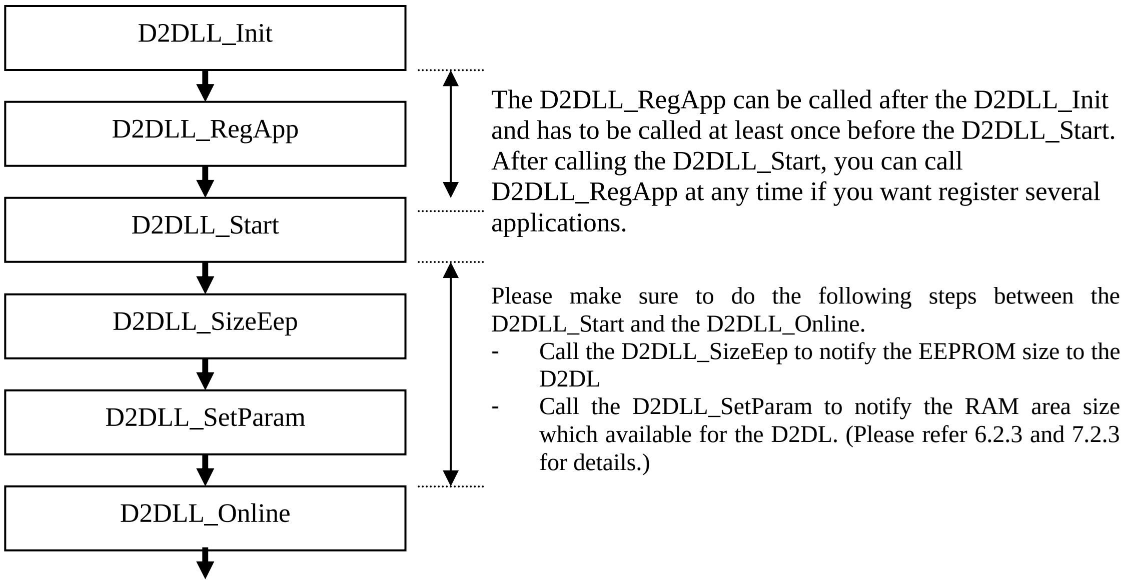
This section describes the necessary information that is common to the Single Task Edition and the Multi Task Edition to develop the user application.

5.1 Initialize Sequence

The following 6 API functions are used for the D2DL initialize and startup sequence.

Function name	Description
D2DLL_Init	Initialize the RTOS parameter for the D2DL
D2DLL_Start	Start the D2DL (for setting the address and the received packet type, etc).
D2DLL_RegApp	Register the application
D2DLL_Online	Enable the PLC communication function
D2DLL_SizeEep	Notify the size of your EEPROM
D2DLL_SetParam	Set the D2DL parameter

Please call functions above in the given order to initialize and to start the D2DL.



The D2DLL_RegApp can be called after the D2DLL_Init and has to be called at least once before the D2DLL_Start. After calling the D2DLL_Start, you can call D2DLL_RegApp at any time if you want register several applications.

Please make sure to do the following steps between the D2DLL_Start and the D2DLL_Online.

- Call the D2DLL_SizeEep to notify the EEPROM size to the D2DL
- Call the D2DLL_SetParam to notify the RAM area size which available for the D2DL. (Please refer 6.2.3 and 7.2.3 for details.)

After calling the D2DLL_Online, the PLC communication (transmission and reception) is available.

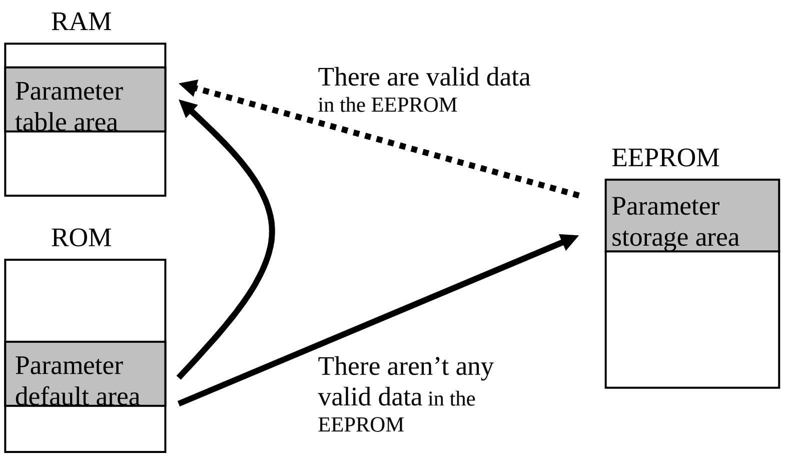
5.2 Handling the internal parameters for the D2DL

The D2DL stores the internal main parameter to EEPROM and manages them. When the D2DL starts up, it lays out these data to the table on the RAM. If the D2DL refers them, it accesses the data on the RAM table. The use application can access the following 12 disclosed parameters. (The disclosed parameters may be added for the future.)

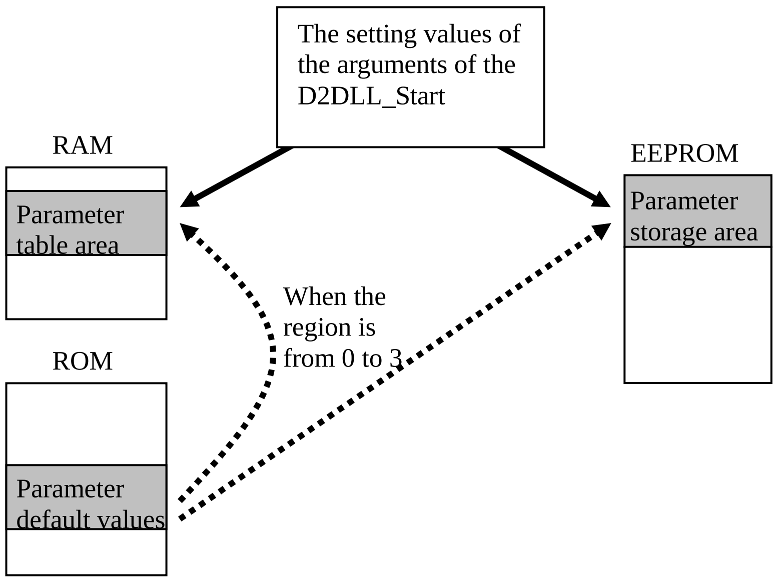
Parameter	Default value
Network ID	0
Node ID	1
Received packet type	All packet types are disabled (it does not receive any packets).
Region	FCC (N.America)
Device Serial Number	All zero
Selection of the repeater function ON/OFF	ON
The retransmission number of ACK service	3
The retransmission number of UNACK service	0
Protocol number	1
Transmission rate	AUTO
Buffer size of the transmission and reception packet. (Please refer 6.2.3 and 7.2.3 for details.)	3584
The size of the specific area for the received packet. (Please refer 6.2.3 and 7.2.3 for details.)	512

The handling of the internal parameters for the D2DL is as follows.

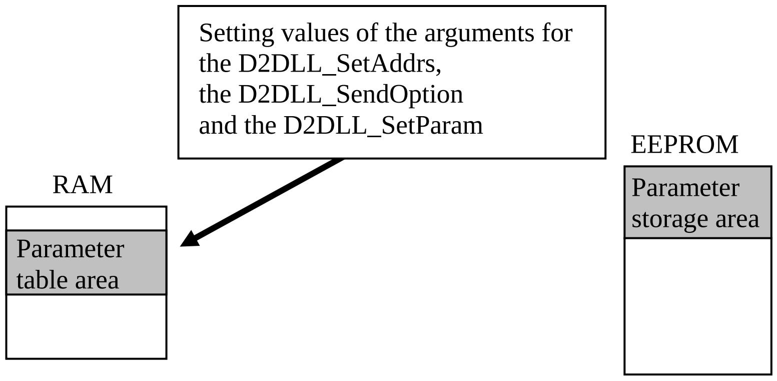
- Calling the D2DLL_Init
  The D2DL accesses the parameter area of the EEPROM. If there are valid data, (they are checked with checksum etc.), the D2DL copies them to the RAM table. If there are no valid data, (for example the EEPROM is blank), the D2DL copies the default data which is stored in the ROM to the table of the RAM and the EEPROM. In this case the default values are the parameters for FCC (U.S.A).



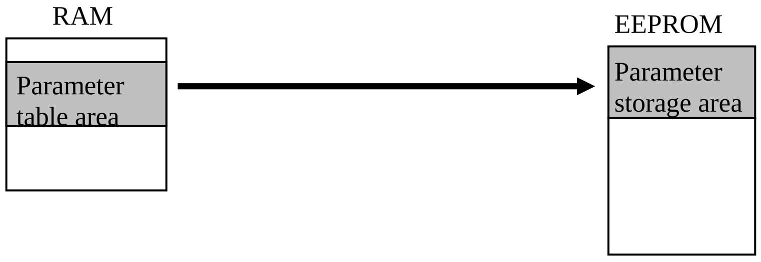
- Calling the D2DLL_Start
  The D2DL sets the parameters which are indicated by the arguments (i.e. selecting network ID, node ID, received packet type, region, device serial number and repeater function ON/OFF) to the RAM table and the EEPROM. If the region is set from 0 to 3, the default values of the region are loaded and then the other parameters are set.


The setting values of the arguments of the D2DLL_Start
RAM
Parameter table area
ROM
Parameter default values
When the region is from 0 to 3
EEPROM
Parameter storage area

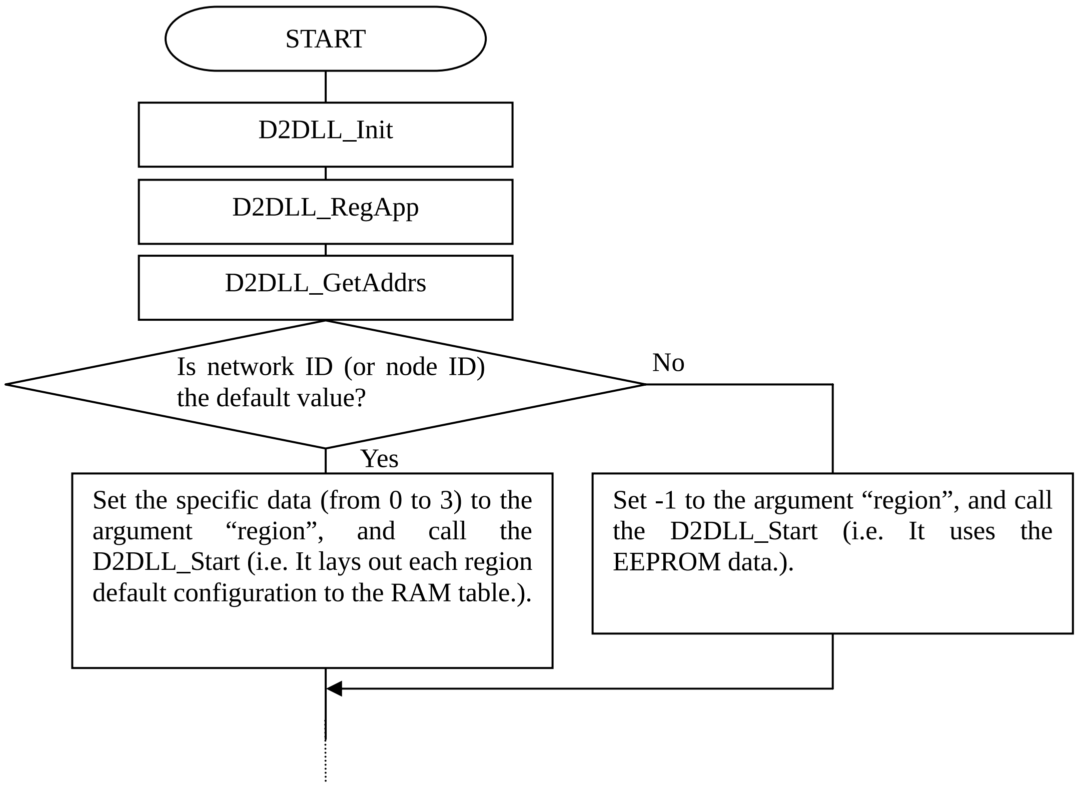
- Calling the D2DLL_SetAddrs, the D2DLL_SendOption and the D2DLL_SetParam
  The parameter values of the RAM can be changed.
  These functions don't change the data of the EEPROM. If you want to store the data of the RAM table, please call the D2DLL_SaveParam as follows.


Setting values of the arguments for the D2DLL_SetAddrs, the D2DLL_SendOption and the D2DLL_SetParam
RAM
Parameter table area
EEPROM
Parameter storage area

- Calling the D2DLL_ SaveParam
  It copies the data of the RAM table to the EEPROM.



If you want to judge whether you load the default data to the EEPROM or use the valid data of the EEPROM before calling the D2DLL_Start, you can judge it by the address values (node ID and network ID).


START
D2DLL_Init
D2DLL_RegApp
D2DLL_GetAddrs
Is network ID (or node ID) the default value?
No
Yes
Set the specific data (from 0 to 3) to the argument "region", and call the D2DLL_Start (i.e. It lays out each region default configuration to the RAM table.).
Set -1 to the argument "region", and call the D2DLL_Start (i.e. It uses the EEPROM data.).

If you have any cases except the address, just feel free to contact us.

5.3 EEPROM interface process

The D2DL opens the source code for EEPROM I/F, therefore you can use any EEPROM. By default an example program which supports the **ATMEL AT24C128** and **AT24C16** EEPROM is included. It uses **SCL2 (pin no.18)** and **SDA2 (pin no.19)** for communication between EEPROM and M16C/6S. Therefore if you use the same specific hardware, you will be able to use the example EEPROM I/F without modification. (Note: Please set the compile option _EEP_128K/ _EEP_16K according to your EEPROM. For more details, please follow the specification described in "D2DL EEPROM I/F SPECIFICATION".)
In other cases, if you use the different hardware, please modify the EEPROM I/F to be fitted to your EEPROM and the communication I/O specification of M16C/6S following the "**D2DL EEPROM I/F Specification**".

5.4 Configuration of the protocol number

The D2DL keeps the coexistence of any products which implements different upper protocols by using the protocol number. The nodes using the same protocol number can communicate with each other, but the nodes which use different protocol number cannot. For example, when there are two nodes which have the same address (network ID and node ID) on the line, they can keep the coexistence if each of them uses a different protocol number. If you want to use this feature, you should set the following protocol number according to your protocol.

Upper protocol	Protocol number
Echonet	0x01
User specific protocol	0x3B

Please contact the following address if you want to use other standard protocol.

E-mail address: plcsupport@renesas.com

You can set the protocol number by calling the D2DLL_SetParam function with the following arguments.

The first argument	The second argument
D2DLL_IDX_PROTOCOL_VER(70)	Protocol number

The example description is as follows.

```
d2dllResult = D2DLL_SetParam(D2DLL_IDX_PROTOCOL_VER, 0x3B );
if( d2dllResult != D2DLL_E_OK    ){
        while(1);
}
```

[Note] Do not set the protocol number except 0x01 and 0x3B in current version.

If you use the user specific protocol number, each product of the different company can communicate with each other because the same protocol number (0x3B) is set. Therefore it is recommended that you implement your measures for avoiding the interference between each product in the user application.
The default value of the protocol number is 0x01. Please refer to the section 5.2 about the handling of the initialization values.

5.5 Notes on using the CNC service

The device serial number (DSN) is used as the address in the CNC service. The DSN is the unique 16bytes value which should be assigned to each node, and it is stored in the EEPROM of each node. The user application in the node that receives a CNC packet can recognize the source device by the source DSN that is included in the received packet.
The DSN should be assigned by the user to avoid overlappings between the CNC packet of the user products. To achieve this necessity, a customer number is assigned to each user which is included in the DSN.

The customer numbers are managed in Renesas Solutions Corp. **Please contact to the following e-mail address if you want to use the CNC service.** Then you will receive an e-mail in response which includes your customer number and its information of implementation.

Email address: plcsupport@renesas.com

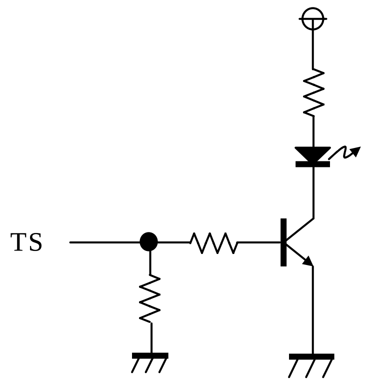
5.6 LED port assignment

The control signal of LED which indicates the packet reception can be assigned to any I/O port. If you implement the following functions in the user application, you can activate the LED on receiving the packet.

void D2DLL_CB_RxLEDon (void)	: LED ON for receiving
void D2DLL_CB_RxLEDoff (void)	: LED OFF for receiving

Please do not write any process to go through quickly except the process of the LED ON/OFF because these functions are called by the interrupt handler. If you do not need the LED for receiving, please make the functions empty.

For the LED transmission control, you can use the “TS” (Pin31) of M16C/6S, so it needs not to be controlled by software. The example circuit for LED control using the TS is shown as follows.



6. Development the User Application (Single Task Edition)

This section describes about the necessary information to develop the user application with the D2DL of the single task edition.

6.1 Example program organization

It is recommended that you develop the user application program based on the D2DL example program.
The project of the example program corresponds with the RENESAS HEW (High-performance Embedded Workshop). When you execute the D2DL setup file, the following folders are created on your PC.

Folder Name	Description
d2dllApp_s	This is a work space folder. There is a HEW work space file (*.hws). If you open and use this file on HEW, you can develop your program based one the example program.
D2dllApp_Vxxxs(- Note)	This is a project folder. The information about this project is stored under this folder.
Debug	There is debug configuration information. The configuration includes build option information, etc. Output file is also stored here after build process.
Release	Not used.
src	There are some source files of the example program.
app	See below.
eeprom	
inc	
lib	

[Note:] ”xxx” is a version number.

Following table shows each file in the src folder. Please develop your application by modifying “UserMain.c”.

Folder Name	File Name	Description	Language	Modifiable
app	UserMain.c	This is an example source file for the User Application Program. The function named Main_WorkerThread is like a function “main()” in a C program. **In your application programming, please modify this function and add your sub-functions**.	C	Yes
	dll_heap.a30	This is a file for the heap area which is used by D2DL library. When you change the size of the heap area, please change this file.	ASM	Yes
	mrtable.a30	This is a file for interrupt vector table. When you add your own interrupt service routine, please add the interrupt vectors in this file.	ASM	Yes
	stack.a30	This is a file for the stack. Please define stack size for user application (i.e. main routine and interrupt).	ASM	Yes
	startup.r30	This is a module file for starting up.	--	No
eeprom	eeprom16C.c	This is a file for EEPROM I/F which includes the function to access to EEPROM. If you use the different hardware with the example program, please modify this file to be fitted to your EEPROM and the communication I/O specification of M16C/6S. (For details, please refer 5.3.)	C	Yes
Inc	D2Dll.h	This is a header file which contains declarations for prototype of D2DL APIs and data types. Please include this file into your C source files for the User Application Program.	C	No
	valType.h	This is a header file which contains declarations for basic data types of each API function. Please include this file before including D2Dll.h into your C source files for User Application.	C	No
lib	d2dll.lib	This is a library file for D2DL.	--	No
	kernel1.lib	This is a library file for MiniRTOS of D2DL.	--	No
	kernel2.lib	This is a library file for MiniRTOS of D2DL.	--	No

6.2 Implementation method

This section describes the necessary information when you develop the program using the D2DL.

6.2.1 Transmission and Reception Sequence

The user application may need the following features.

- Getting the transmission result of the requested packet
- Getting the received packet

The D2DL provides the next two methods to support above features.

- The user application can get the information via callback function that is called by D2DL.
- The user application can get the information by calling the API function.

The user application can get the information from the D2DL by using either one of both methods.
This section describes each method.

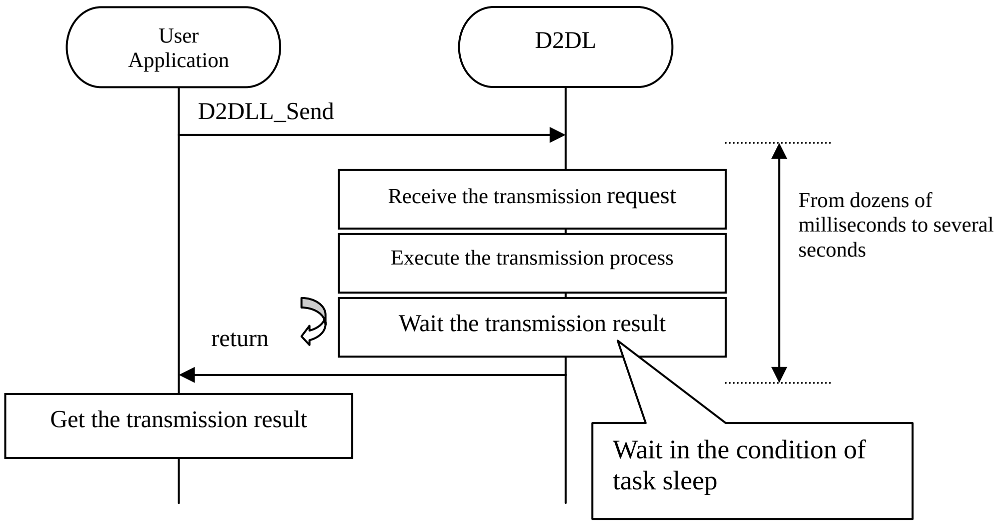
[Notes]
When you use the single task edition, we recommend using the callback functions. If the user application does not use the callback function and it calls the API function by the method which takes wait time, the process waits in the function, and it does not return until when the communication is completed.

6.2.1.1 Getting the packet transmission result

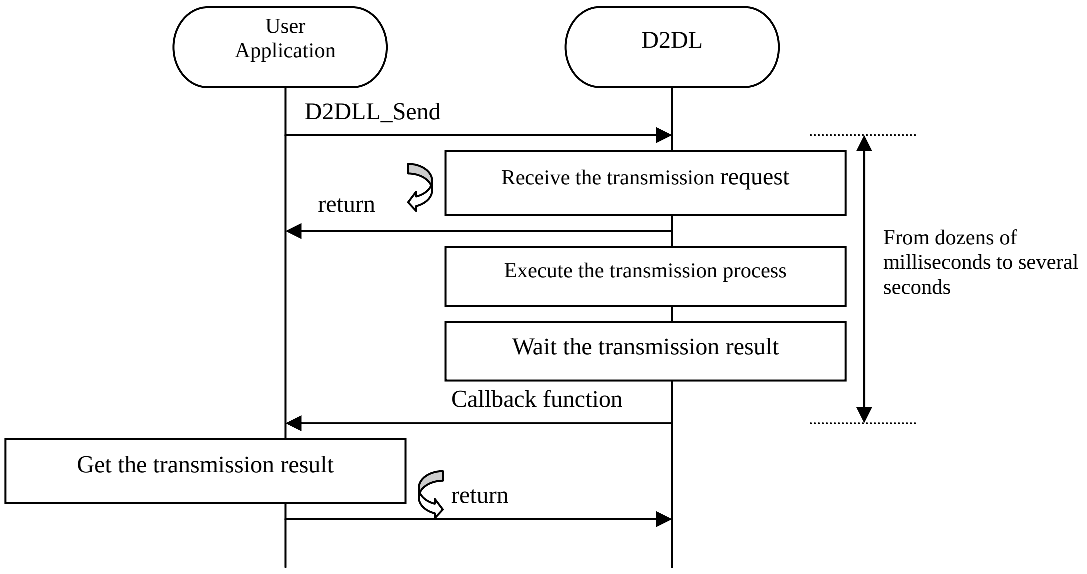
The method to get the transmission result is determined by the 3rd argument of the D2DLL_RegApp function.

- Case: The 3rd argument of the D2DLL_RegApp is NULL

  The transmission result is passed by the return value of the packet transmission request API function (i.e. the D2DLL_Send etc.). Therefore the transmission function does not return until the transmission result has determined. It takes dozens of milliseconds up to several seconds to determine the transmission result, according to the condition of the line and the destination node. The task which calls the D2DLL_Send function has been in the sleep status until the transmission result has determined. In this period, the other task runs in the Multi Task Edition, but in the Single Task Edition the user task stops.


User Application
D2DL
D2DLL_Send
Receive the transmission request
Execute the transmission process
Wait the transmission result
return
From dozens of milliseconds to several seconds
Get the transmission result
Wait in the condition of task sleep

- Case: The 3rd argument of the D2DLL_RegApp is not NULL
  The API transmission function (i.e. the D2DLL_Send etc.) terminates as soon as the transmission request is received in the D2DL. (The wait time does not occur.) After that, when the transmission result is determined, the D2DL calls the callback function which start address is the value of the 3rd argument of the D2DLL_RegApp, and passes the transmission result to the user application.

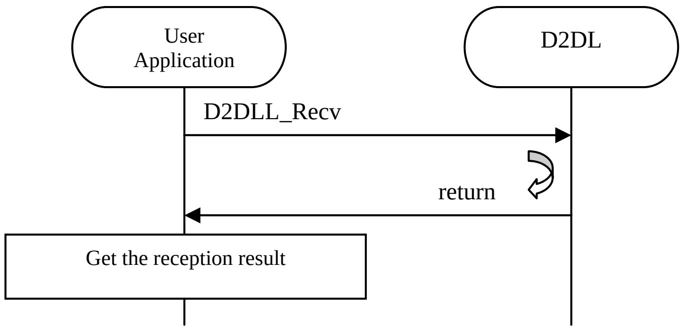

User Application
D2DL
D2DLL_Send
Receive the transmission request
return
Execute the transmission process
Wait the transmission result
From dozens of milliseconds to several seconds
Callback function
Get the transmission result
return

6.2.1.2 Getting the received packet

The method to get the received packet is determined by the 2nd argument of the D2DLL_RegApp function.

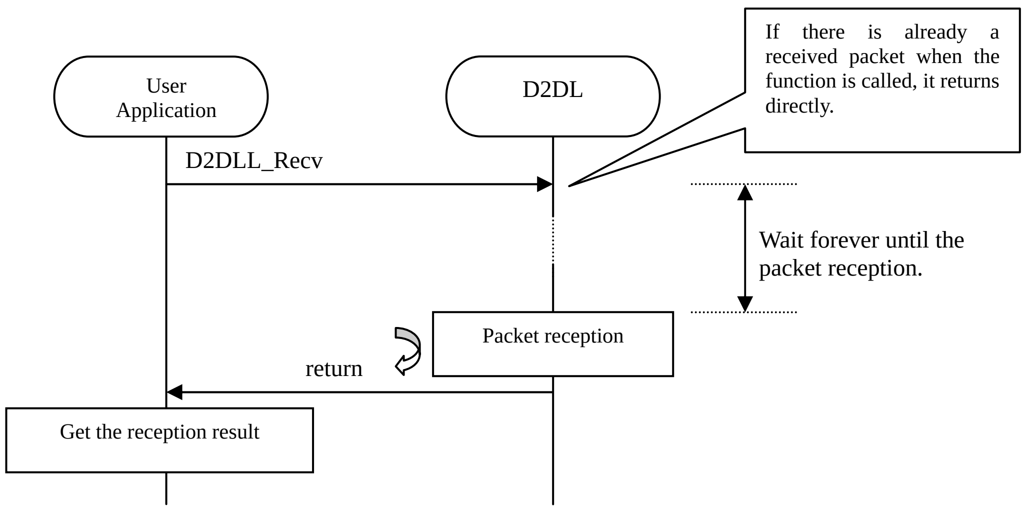
- Case: The 2nd argument of the D2DLL_RegApp is NULL
  The reception API function D2DLL_Recv gets the received packet. The reception waiting time can be set by the argument "rcvTimeout" of the D2DLL_Recv function.

  - If the rcvTimeout is D2DLL_NOWAIT(0x0000),
    The D2DLL_Recv function is terminated whether there is a received packet or not when it is called.

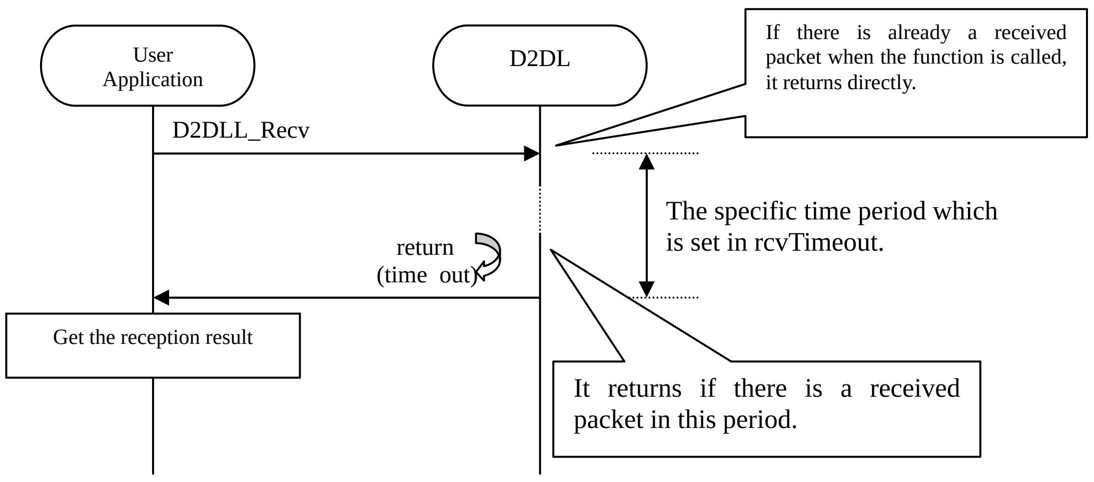


  - If the rcvTimeout is D2DLL_INFINITE (0xFFFF),
    If there is not a received packet, the D2DLL_Recv waits of the packet reception internally forever. When the packet is received, the function is terminated. If there is a received packet when the function is called, it is terminated immediatley. While waiting for the received packet, the task which calls the D2DLL_Recv function has been in the

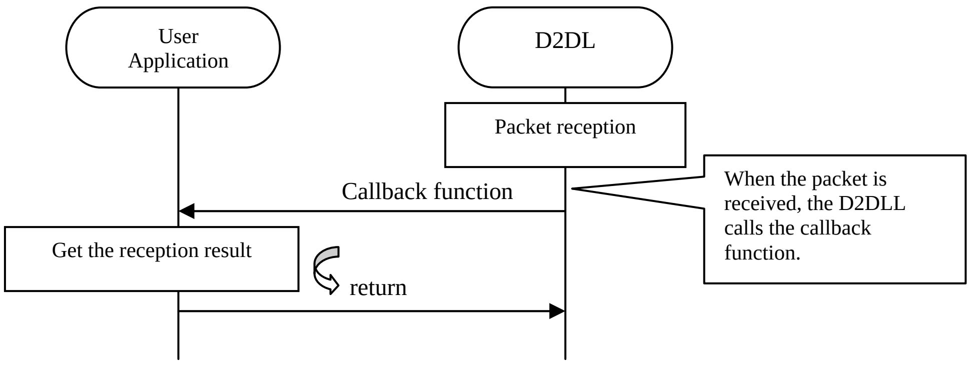
sleep status. In this period, the other task runs in the Multi Task Edition, but in the Single Task Edition the user task stops.


User Application
D2DL
If there is already a received packet when the function is called, it returns directly.
D2DLL_Recv
Wait forever until the packet reception.
Packet reception
return
Get the reception result

- If the rcvTimeout is neither D2DLL_NOWAIT(0x0000) nor D2DLL_INFINITE (0xFFFF),
  If there is a packet when the D2DLL_Recv is called, it is terminated immediatley. If there is not a received packet, it waits of the packet reception for the specific period which is set in rcvTimeout (Unit: msec), and it is terminated on the packet reception. If it does not receive a packet until the specific time is expired, then the function is terminated. While waiting for the received packet, the task which calls the D2DLL_Recv function has been in the sleep status. In this period, the other task is running in the Multi Task Edition, but in the Single Task Edition the user task stops.


User Application
D2DL
If there is already a received packet when the function is called, it returns directly.
D2DLL_Recv
The specific time period which is set in rcvTimeout.
return
(time out)
Get the reception result
It returns if there is a received packet in this period.

- Case: The 2nd argument of the D2DLL_RegApp is not NULL
  The received packet is passed to the user application by the callback function from the D2DL. In this case, the D2DLL_Recv function is not needed.



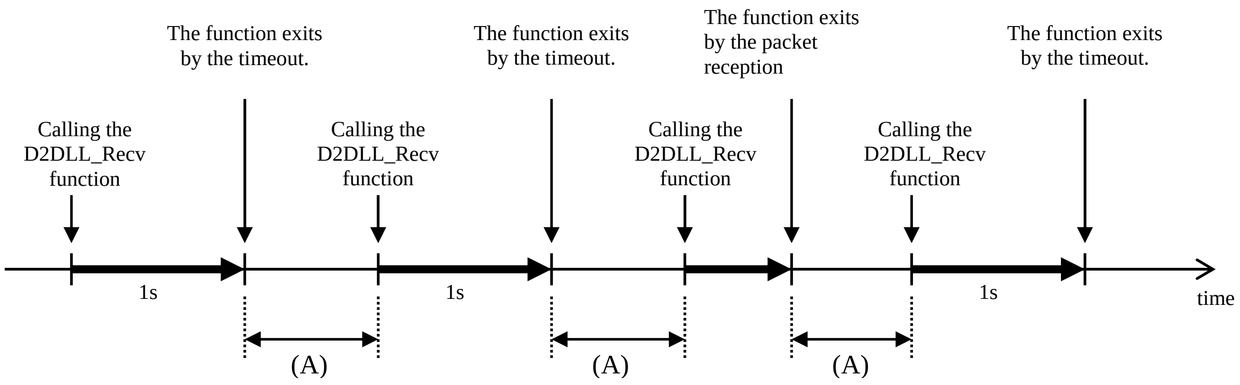
In the environment in which there are some application (port) on the D2DL, the function which is registered in each application is called back in case of the packet reception callback function. However in case of the D2DLL_Recv function, it cannot assign the received packet to each destination application. If you use the D2DLL_Recv function in the multiple application environment, please implement the assignment process for each application by checking the dstPortId (destination port ID) in the received data.

6.2.1.3 Restrictions on using the reception function

When the user application receives the packet by the packet reception function D2DLL_Recv, not the callback function, the D2DLL_Recv function should be called at short intervals as the following.

Region	Maximum call interval
U.S.A (FCC)	50ms
Japan (ARIB)	90ms
Europe (CENELEC A/B)	120ms

The timeout interval which is set in the argument "rcvTimeout" of the D2DLL_Recv function is not included in times shown above. An example when timeout is one second is shown as below. **Please implement the following (A) interval not to exceed the maximum call interval which is shown in the table above.**


The function exits by the timeout.
The function exits by the timeout.
The function exits by the packet reception
The function exits by the timeout.
Calling the D2DLL_Recv function
Calling the D2DLL_Recv function
Calling the D2DLL_Recv function
Calling the D2DLL_Recv function
1s
1s
1s
time
(A)
(A)
(A)

6.2.2 Addition of the Interrupt Handler

The function for interrupt service routine can be written in C. The method of addition is different between Watchdog Timer (hereafter called WDT) interrupt and others. So please refer to the following.
Note that it is not available to use the D2DL API in the interrupt process function.

- Addition of the interrupt process function
  Please declare the return value and the argument as "void". Do not declare as static.
  Please describe the following at top of the file which defines the interrupt function.

[In case of interrupt excluding WDT:]

```
_asm("__MR_IntEntry      .MACRO      ¥n.ENDM ");
#pragma INTHANDLER           Interrupt Function Name
```

[In case of WDT:]

```
#pragma INTERRUPT       Interrupt Function Name
```

Example of description for interrupt excluding WDT:

```
_asm("__MR_IntEntry   .MACRO   ¥n.ENDM ");
#pragma   INTHANDLER   inthand
void inthand(void)
{
     /* process*/
}
```

If you enable the multiple interrupt, please add "/E" after INTHANDLER or INTERRUPT.

- Addition to "mrtable.a30"
  Add your interrupt process function to the interrupt vector. It can be set below line number 171 of "mrtable.a30".

  [In case of interrupt excluding WDT]
  Please set each vector as the same as the following gray shaded lines <<1>> and <<2>>.

  <<1>>Please add the following 3 lines. Add the vector number to [Vector Number], and the interrupt function name to [Interrupt Function Name]("[]" is not necessary).
  The "()" is not necessary for function name, and please attach "_" at top of the name.
  <<2>>Please change the dummy function name "__SYS_DMY_INH" to the [Interrupt Function Name] ("[]" is not necessary).

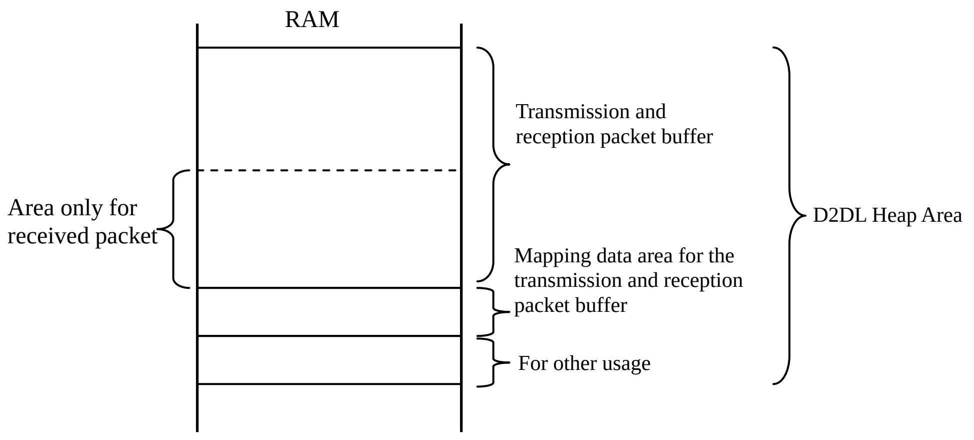
```
; +-----------------------------------------------------------+
; |             mr30 vector table                  |
; +-----------------------------------------------------------+
                .SECTION    INTERRUPT_VECTOR
;
        .GLB  __INTOS_8             ;D2dll_PhyRx_PhyUsartISR()
__INTOS_8.                 EQU   1
                .GLB  _D2dll_PhyRx_PhyUsartISR
        .GLB  __INTOS_[ Vector Number]      ; [Interrupt Function Name]              << 1 >>
__INTOS_[ Vector Number]    .EQU  1
                .GLB  [Interrupt Function Name]
                        :
                        :
                .LWORD              __SYS_DMY_INH           ;vector7
                .LWORD              _D2dll_PhyRx_PhyUsartISR        ;vector8
                .LWORD              __SYS_DMY_INH           ;vector9
                .LWORD              [Interrupt Function Name]        ;vector[Vector Number]<< 2 >>
                .LWORD              __SYS_DMY_INH           ;vector11
                        :
                        :
```

[In case of WDT interrupt]
Please set each interrupt function as the same as the following gray shaded lines <<1>> and <<2>>.
<<1>> Please define the interrupt function name to [Interrupt Function Name] ("[]" is not necessary).
The "()" is not necessary for the function name, and please attach "_" at top of the name.
<<2>> Please change the dummy function name "__SYS_DMY_INH" on line for vector 252 to the [Interrupt Function Name] ("[]" is not necessary).

```
; +------------------------------------------------------------+
; |               mr30 vector table                    |
; +------------------------------------------------------------+
                .SECTION    INTERRUPT_VECTOR
;
        .GLB   __INTOS_8             ;D2dll_PhyRx_PhyUsartISR()
__INTOS_8.                   EQU   1
                .GLB  _D2dll_PhyRx_PhyUsartISR
                        :
                .GLB  [Interrupt Function Name]                                        << 1 >>
                        :
                        :
                .LWORD                __SYS_DMY_INH           ;vector250
                .LWORD                __SYS_DMY_INH           ;vector251
                .LWORD                [Interrupt Function Name]               ;vector252             << 2 >>
                .LWORD                __SYS_DMY_INH           ;vector253
                        :
                        :
```

6.2.3 Configuration of the D2DL Heap Area

The D2DL uses RAM area for the transmission and reception packet buffer, and other usages. (This area is called "the heap area" in this document.) The size of the heap area is configurable by the user and the allocation is as follows:


RAM
Transmission and reception packet buffer
Area only for received packet
D2DL Heap Area
Mapping data area for the transmission and reception packet buffer
For other usage

The packets which are requested to be transmitted by the user application and the packets which are received by PHY are temporarily stored in the "transmission and reception packet buffer". If the upper layer requests transmission continuously, for example, the transmission waiting packet occupies the buffer, so the user application cannot receive any packets. Therefore the "area only for received packet" is the area which is always allocated in the transmission and reception packet buffer for the received packet usage to avoid this condition.
"Mapping data area for the transmission and reception packet buffer" is the area which tracks the data location on the transmission and reception packet buffer.

Please set the size of these areas according to your system because they depend on the target system specification, such as the maximum payload size of the transmission and reception packet and the transmission and reception frequency.

The following calculated value gives an indication of the size which area used by one packet (transmission and reception packet) in the transmission and reception packet buffer.

"The payload size of the packet" + 17(header size) + 101 (byte)
[Notes] You do not need to consider about the packet fragmentation on this calculation. For example, the 400 bytes payload packet occupies "400 + 17 + 101 = 518 bytes".

The size of the data area for the transmission and reception packet buffer mapping is calculated as follows.
The transmission and reception packet buffer size (bytes) / 32

[Note] Round up the remainder.

“For other usage” area is fixed 284 bytes.

You should allocate the total size of the area which is added to each RAM area, and set the size to the D2DL. The method of setting is as follows.

The heap area size is set by the following two methods.

- The parameter which is written in “dll_heap.a30”
- The parameters which are set by the function “D2DLL_SetParam”

Both of them should be set. First, it describes the configuration by “dll_heap.a30”.

Please set the size of your heap area to the following gray shaded line in “dll_heap.a30” in hexadecimal. Do not change other item except this value in this file.

```
        .GLB D2DLL_HEAP_AREA, __d2dll_heap_top
D2DLL_HEAP_AREA         .EQU 600H           ; 1.5K
;-------------------------------------------------------------
; heap section (to be used by d2dll)
;-------------------------------------------------------------
        .GLB _g_d2dll_StartOfHeap
        .GLB _g_d2dll_EndOfHeap
        .section        d2dll_heap,DATA,ALIGN
        _g_d2dll_StartOfHeap:           .BLKW     1
        _g_d2dll_EndOfHeap:             .BLKW     1
__d2dll_heap_top:
        .BLKB                D2DLL_HEAP_AREA
        .END
```

In the start up file “startup.r30”, the “_g_d2dll_StartOfHeap” is set to the top address of the heap area and the “_g_d2dll_EndOfHeap” is set to the end address of the heap area automatically. These values are passed to the D2DL as the return value of the function “HALMemMngS_GetFreeRAMStart” and “HALMemMngS_GetFreeRAMEnd”. Therefore you should implement these functions in the user application. (You can use the source code of the example program without any changes.)

[The function for getting the start point]

```
unsigned HALMemMngS_GetFreeRAMStart( void )
```

[The function for getting the end point]

```
unsigned HALMemMngS_GetFreeRAMEnd( void )
```

Next, it describes the configuration by the function “D2DLL_SetParam”.

Please set the size of your heap area by the following argument of the function “D2DLL_SetParam”.

The first argument	The second argument
D2DLL_IDX_DLL_MEMORY(71)	The size of the” transmission and reception packet buffer” (bytes)
D2DLL_IDX_RXRES_MEMORY (77)	The size of the “area only for received packet” (bytes)

The example source code is as follows.

```
d2dllResult = D2DLL_SetParam( D2DLL_IDX_DLL_MEMORY, 916 );
if( d2dllResult != D2DLL_E_OK    ){
            while(1);
}
d2dllResult = D2DLL_SetParam( D2DLL_IDX_RXRES_MEMORY, 458 );
if( d2dllResult != D2DLL_E_OK    ){
        while(1);
}
```

[Note] These configurations should be set between calling the "D2DLL_Start" and the "D2DLL_Online" in the D2DL initialization sequence.

Please refer to the example program for more detailed implementations.

6.2.4 Configuration of the stack size

The default stack size of the D2DL is as follows.

- The system stack (which is used by the MiniRTOS and the interrupt routine in the D2DL) is 330 bytes.
- The stack of the user application main loop is 280 bytes. (It is used by the example program.)

The method to change the stack size is as follows.

The configuration of the stack size is in the "stack.a30" file.
Please calculate each stack size for example, by the stack size calculation program "StkViewer" and set them to the following gray shaded lines<<1>><<2>>.

<<1>>Set the system stack size in hexadecimal.
The interrupt process of the user application also uses the system stack.
Do not set smaller value than the default.

<<2>>Set the stack size which is used by the main routine "Main_WorkerThread()"of the user application in hexadecimal.

Do no changes except these values in this file.

```
        .GLB __SYS_STACK_SIZ, __Main_stack, __Sys_Sp
__SYS_STACK_SIZ    .EQU        014aH                    << 1 >>
__Main_stack     .EQU        0118H                    << 2 >>
; +-------------------------------------+
; |    Stack Area         |
; +-------------------------------------+
        .section      stack,DATA,ALIGN
        .BLKB              __SYS_STACK_SIZ
        .align
__Sys_Sp:
        .BLKB              __Main_stack
        .BLKB              0208H
        .END
```

6.3 Notes

6.3.1 Influence of the D2DL task and its interrupt process on the user application

During the main function “Main_WorkerThread” of the user application or the new interrupt functions which are added by the user is running, **the interrupt process for the D2DL** may start up and then **the tasks in the D2DL library** run circumstantially.
In this case, **the user application pauses** and **the process of the D2DL library starts to run**. The occurrence frequency of this D2DL interrupt process (which includes the D2DL task process) depends on the amount of PLC packets.

6.3.2 Notes on the interrupt

- If the processing time of the interrupt process of the user application meets one of the following conditions, we recommend enabling multiple interrupt with a priority level of 3 or more. Without enabling this, some packets might be lost.

  Condition 1 : the processing time of one interrupt process of the user application is 100 us or more, and its interrupt may occurs in a 2.4 ms interval or less

  Condition 2 : the processing time of one interrupt process of the user application is longer than 500 us

- The interrupt processing time of the D2DL and its occurrence timing are as shown.

Trigger of the Interrupt	Function Name	Processing Time	Occurrence Timing
SI/O4	D2dll_PhyRx_PhyUsartISR	20 us – 500 us	The transmission and reception on the PLC. (In other case, it may occur at random times by the reception of the noise.)
INT0	D2dll_Phy_ISR	20 us – 120 us	
TA0	D2dll_Timer_ISR	30 us – 90 us	
TA1	D2dll_Timer1_ISR	10 us or less	

[Notes] The SI/O4 interrupt process takes about 500 us only in case of the reception for the top byte of the packet. In other case, the process takes about 20 us.

- To disable the interrupt in the user application, please select one of the following methods.
  I. Disable the interrupt by the operation of the interrupt enable flag (I flag) and the processor interrupt priority level (IPL).
  Please make sure to use the interrupt disabling and enabling in pairs.
  II. Change the interrupt priority level (from ILVL2 to 0) to disable.
  Please clear the I flag before and after the operation of the interrupt priority level (from ILVL2 to 0).

- The D2DL reserves the interrupt number from 32 to 47 of the INT instruction. Please use the interrupt number except from 32 to 47 when you use the software interrupt in the user application.

- Register Bank Usage
  It is not available to change the register bank.

6.3.3 Notes on the debug

- You can set the breakpoints up to two at the same time (in case of using the KD30).
- Do not use the DBC interrupt because it is the specific interrupt for the developer support tool.
- It is not available to show the source code of the startup file (startup.r30) during debugging by means of the KD30 or the PD30 because it is provided only in r30 format.

7. Development of the User Application (Multi Task Edition)

This section describes the necessary information to develop the user application using the multi task edition of the D2DL.

7.1 Example program organization

It is recommended that to develop the user application program based on the D2DL example program.
The project of the example program corresponds with the RENESAS HEW (High-performance Embedded Workshop). When executing the D2DL setup file, the following folders are created on the PC.

Folder Name	Description
d2dllApp_m	This is a work space folder where the HEW work space file (*.hws) is located. The development of own programs should be based on the example program using this work space file in the HEW.
D2dllApp_Vxxxs(- Note)	This is a project file where the project information are stored.
Debug	The debug configuration information, including the build option information, etc. are stored here. After the build process the Output file is also located here.
Release	Not used.
src	There are some source files of the example program.
app	See below.
eeprom	
inc	
lib	

[Note:] "xxx" is a version number.

Following table shows each file in the src folder. Please develop your application by modifying "UserMain.c" and "D2DLLApp.cfg".

Folder Name	File Name	Description	Language	Modifiable
app	UserMain.c	This is an example source file for the User Application Program. The function named Main_WorkerThread is like a function "main()" in a C program. **Programming own application, please modify this function and add sub-functions here**.	C	Yes
	crt0mr.a30	This is a source file of the startup process and also for the heap area which is used by D2DL library. To change the size of the heap area, please change this file and do not change the other configurations.	ASM	Yes
	c_sec.inc	This is a section definition file. Do not change.	ASM	Yes
	D2DLLApp.cfg	This is a configuration file for the real time OS MR30. It defines the system, tasks, flags, semaphores and so on. Please add the definition for the user application after checking the notes. (Please refer to the section 7.2.4 for details.)	--	Partly No
eeprom	eeprom16C.c	This is a file for EEPROM I/F which includes the function to access the EEPROM. Use the example program in combination with different hardware, please modify this file to be fitted to the corresponding EEPROM and the communication I/O specification of the M16C/6S. (For details, please refer to 5.3.)	C	Yes
Inc	D2Dll.h	This is a header file which contains prototype declarations of the D2DL APIs and data types. Please include this file into the C source files for the User Application Program.	C	No
	valType.h	This is a header file which contains basic data type declarations. Please include this file before including D2Dll.h into the C source files for the User Application.	C	No
lib	d2dll.lib	This is a library file of D2DL.	--	No
	d2dll.mrc	This is a system call file which includes the RTOS system call used in the D2DL library.	--	No

7.2 Implementation method

This section describes the necessary information required to develop programs using the D2DL.

7.2.1 Transmission and Reception Sequence

The user application may need the following features.

- Getting the transmission result of the requested packet
- Getting the received packet

The D2DL provides the next two methods to support above features.

- The user application can get the information via callback function that is called by the D2DL.
- The user application can get the information by calling the API function.

The user application can get the information from the D2DL by using either one of two methods.
This section describes each method.
When there is a user application task, it runs as the single task edition.

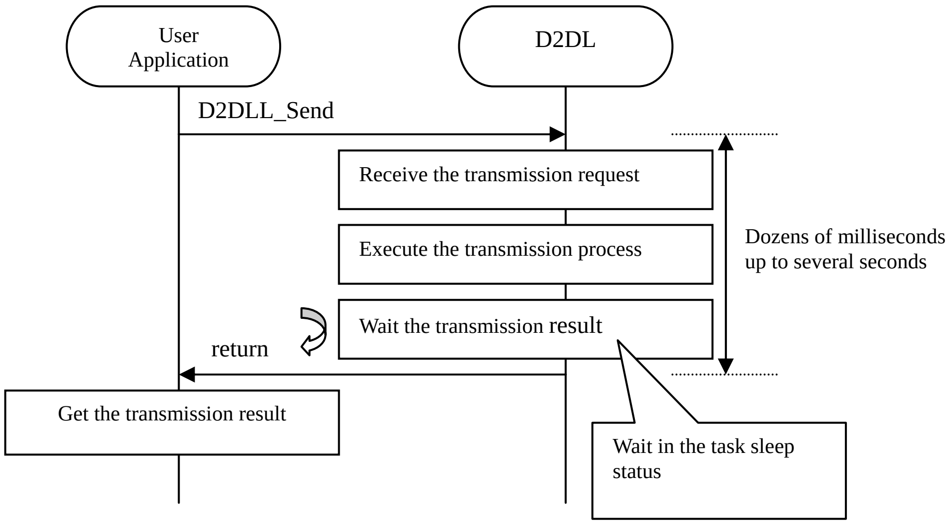
7.2.1.1 Getting the packet transmission result

The method to get the transmission result is determined by the 3rd argument of the D2DLL_RegApp function.

- Case: The 3rd argument of the D2DLL_RegApp is NULL

  The transmission result is passed by the return value of the packet transmission request API function (i.e. the D2DLL_Send etc.). Therefore the transmission function does not return until the transmission result has been determined. It may take dozens of milliseconds up to several seconds to determine the transmission result, according to the condition of the line and the destination node. The task which calls the D2DLL_Send function has been in the sleep status until the transmission result has determined. In this period, the other task runs. When the transmission result is determined, the user task runs again. The priority of the user task should be lower than the internal tasks of the D2DL.

  Multiple tasks can call the packet transmission request API function at the same time. However, until the user task which calls the function first gets the transmission result, other tasks are set to the WAIT state by the semaphore function. Thus the other transmission process is not accepted or executed from the other task even if the first user task is in the sleep status waiting for the transmission result.


User Application
D2DL
D2DLL_Send
Receive the transmission request
Execute the transmission process
Wait the transmission result
Dozens of milliseconds up to several seconds
return
Get the transmission result
Wait in the task sleep status

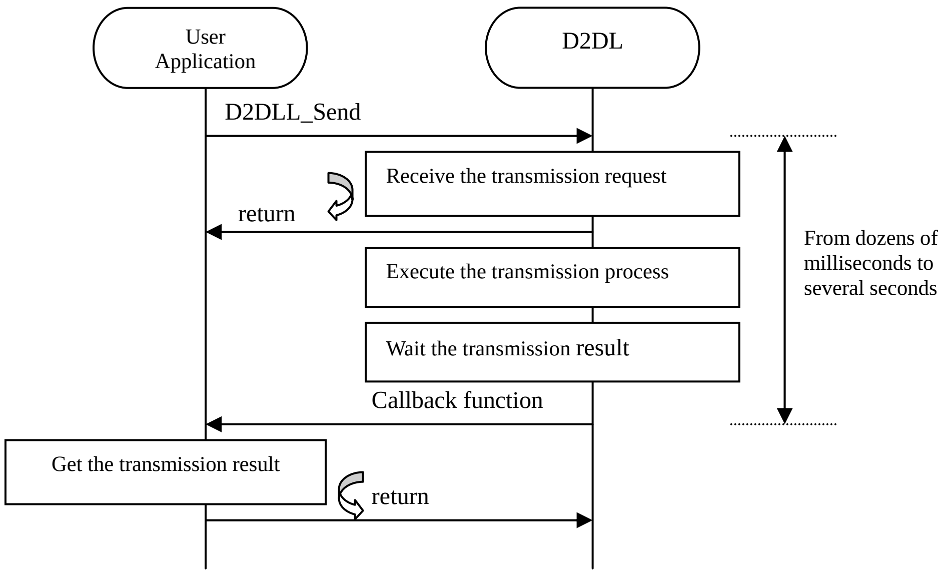
- Case: The 3rd argument of the D2DLL_RegApp is not NULL

  The API function for transmission (i.e. the D2DLL_Send etc.) terminates as soon as the transmission request is received in the D2DL. (The wait time does not occur.) After the transmission result is determined, the D2DL calls the callback function which start address is the value of the 3rd argument of the D2DLL_RegApp, and passes the transmission result on to the user application.

  Multiple tasks can call the packet transmission request API function at the same time. In this case, the callback function should be devised because it is necessary which user task should be passed the transmission result. For example, you can identify which user task should be passed the transmission result by using the member “sessionTag” in the transmission API structure. The example cases are as follows.

Example 1: The multiple user tasks use the same port number for transmission.
Example 2: The multiple user tasks use the different port number for transmission, but use the same callback function.

In these cases, you can identify the task to pass the transmission result by setting the unique “sessionTag” to each user task, i.e. including the user task ID to a part of the “sessionTag” etc.


User Application
D2DL
D2DLL_Send
Receive the transmission request
return
Execute the transmission process
Wait the transmission result
Callback function
From dozens of milliseconds to several seconds
Get the transmission result
return

7.2.1.2 Getting the received packet

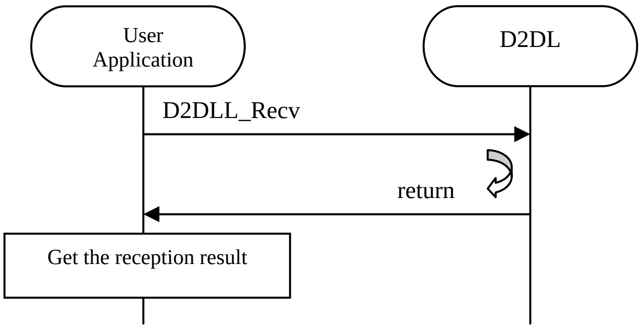
The method to get the received packet is determined by the 2nd argument of the D2DLL_RegApp function.

- Case: The 2nd argument of the D2DLL_RegApp is NULL

  The reception API function D2DLL_Recv gets the received packet. The reception waiting time can be set by the argument "rcvTimeout" of the D2DLL_Recv function.

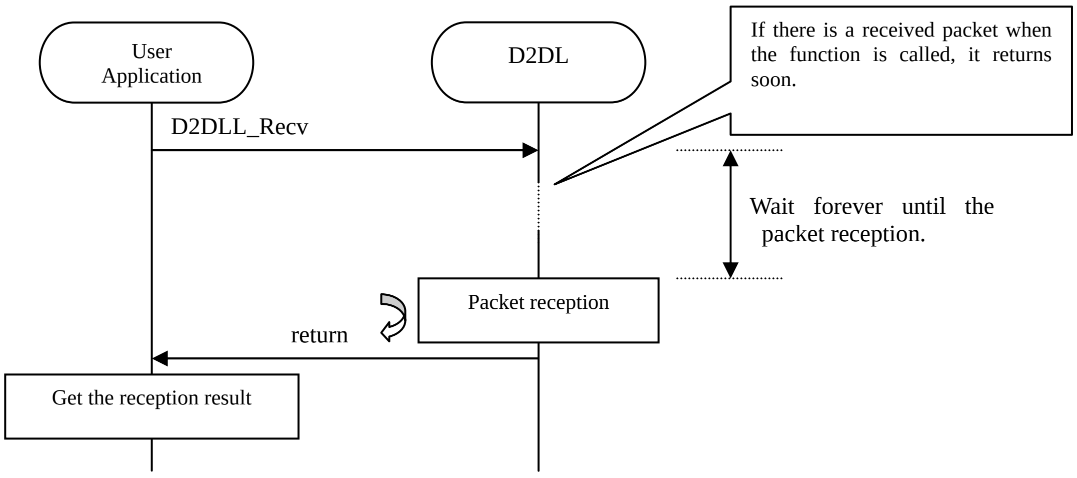
  - If the rcvTimeout is D2DLL_NOWAIT(0x0000),

    The D2DLL_Recv function is terminated whether there is a received packet or not when it is called.



  - If the rcvTimeout is D2DLL_INFINITE (0xFFFF),

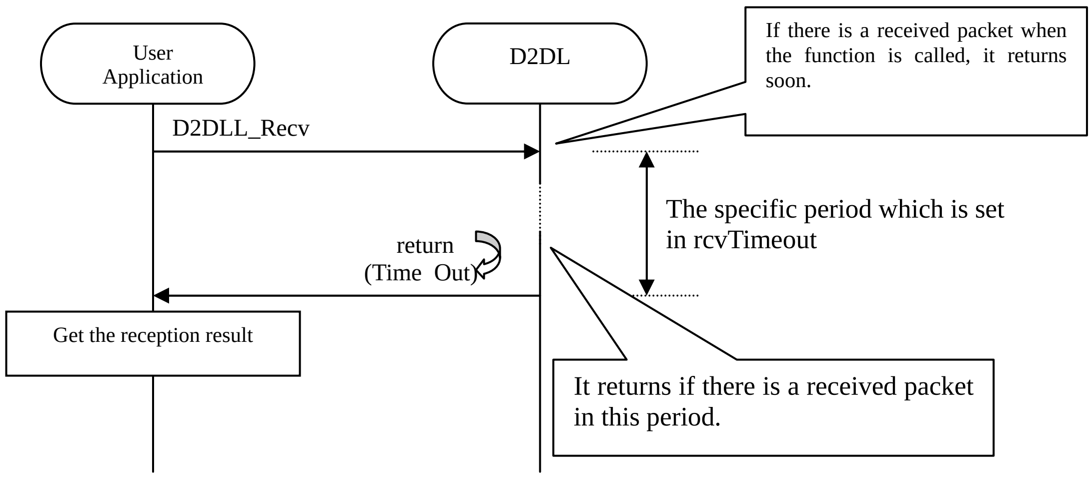
    If there is not a received packet, the D2DLL_Recv waits of the packet reception internally forever. When the packet is received, the function is terminated. If there is a received packet when the function is called, it is terminated soon. While waiting for the received packet, the task which calls the D2DLL_Recv function has been in the sleep status. In this period, the other task runs. When the packet is received, the user task runs again. The priority of the user task should be lower than the internal tasks of the D2DL. Multiple tasks can call the packet reception API function at the same time. However until the user task which calls the function first gets the reception result, and the other tasks are set to the WAIT state by the semaphore function. So the other reception process is not accepted or executed from the other task even if the first user task is in the sleep status waiting for the packet reception.


User Application
D2DL
If there is a received packet when the function is called, it returns soon.
D2DLL_Recv
Wait forever until the packet reception.
Packet reception
return
Get the reception result

- If the rcvTimeout is neither D2DLL_NOWAIT (0x0000) nor D2DLL_INFINITE (0xFFFF),

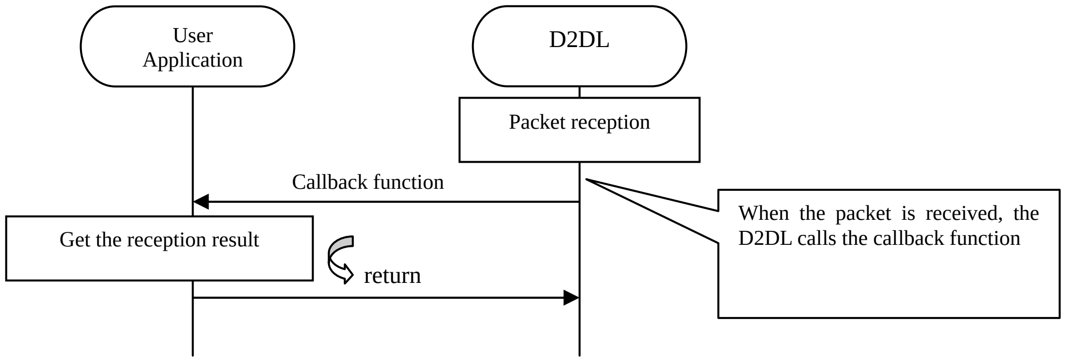
  If there is a packet when the D2DLL_Recv is called, it is terminated soon. If there is no received packet, it is waiting for the packet reception for the specific period which is set in rcvTimeout (Unit: msec). When the packet is received, the function is terminated. If it does not receive a packet while the specific time is expired, then the function is terminated. While waiting the received packet, the task which calls the D2DLL_Recv function has been in the sleep status. As mentioned above for D2DLL_INFINITE (0xFFFF), in this period, the other task runs. When the packet is received or the time-out is indicated, the user task runs again. The priority of the user task should be lower than the internal tasks of the D2DL.

  If the multiple tasks call the reception function at the same time, it is the same sequence.


User Application
D2DL
If there is a received packet when the function is called, it returns soon.
D2DLL_Recv
The specific period which is set in rcvTimeout
return (Time Out)
Get the reception result
It returns if there is a received packet in this period.

- Case: The 2nd argument of the D2DLL_RegApp is not NULL

  The received packet is passed to the user application by the callback function from the D2DL. In this case, the D2DLL_Recv function is not needed. You can pass the reception result on to the multiple tasks by devising the packet reception callback function. For example, when a callback function is used by the different port numbers for the packet reception, you can identify the user task to pass the reception result by using the member "dstPortId" of the structure for the reception API.



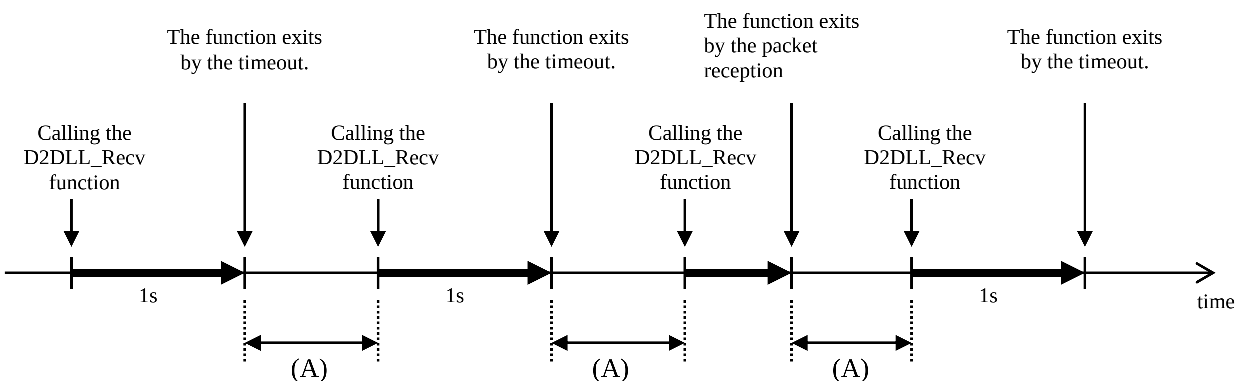
In the environment in which there are some application (port) on the D2DL, the function which is registered in each application is called back in case of the packet reception callback function. However in case of the D2DLL_Recv function, it cannot distribute the received packet to each destination application. If you use the D2DLL_Recv function in the multiple application environment, please implement the assignment process for each application by checking the dstPortId (destination port ID) in the received data.

7.2.1.3 Restrictions on using the reception function

When the user application receives the packet by the packet reception function D2DLL_Recv, not the callback function, the D2DLL_Recv function should be called at short intervals as the following.

Region	Maximum call interval
U.S.A(FCC)	50ms
Japan(ARIB)	90ms
Europe(CENELEC A/B)	120ms

The timeout interval which is set in the argument "rcvTimeout" of the D2DLL_Recv function is not included in times shown above. An example when timeout is one second is shown as below. **Please implement the following (A) interval not to exceed the maximum call interval which is shown in the table above.**


The function exits by the timeout.
The function exits by the timeout.
The function exits by the packet reception
The function exits by the timeout.
Calling the D2DLL_Recv function
Calling the D2DLL_Recv function
Calling the D2DLL_Recv function
Calling the D2DLL_Recv function
1s
1s
1s
time
(A)
(A)
(A)

7.2.1.4 The example of using the callback function

There are two methods to pass the transmission and reception result to the user task by using the callback function.

- <u>By using the MR30 functions:</u>
  The callback function passes the transmission and reception result on to the user task which is waiting for the indication by using the task feature (sleep and wake up), mail feature (waiting the reception and the transmission) or event flag feature (wait and set the flag).
  The following example uses the task operation (sleep and wake up). The sleep task feature can be replaced by the mail reception feature or the waiting for the event flag, and the wake up task feature can be replaced by the mail transmission feature or setting the event flag.

```
[ User task process ]
void UserTask1(void)
{
      :
 errStatus = D2DLL_Send( &sndParam );  /* Transmission request
                                           using the callback function */
 if( errStatus != D2DLL_E_OK ){
      :
 }
 /* wait CB */
 slp_tsk();                            /* Waiting for the indication
                                           of the callback function */
      :
 if (sendStatus != D2DLL_E_OK)         /* Check the result
                                           of the callback function */

      :
}
```

```
[ Callback function process ]
void SendResult_CB( uint16 sessionTag, sint16 sndResult )
{
     :
 sendStatus = sndResult;      /* Get the transmission result */
 wup_tsk(ID_UserTask1);       /* Wakeup the user task */
     :
}
```

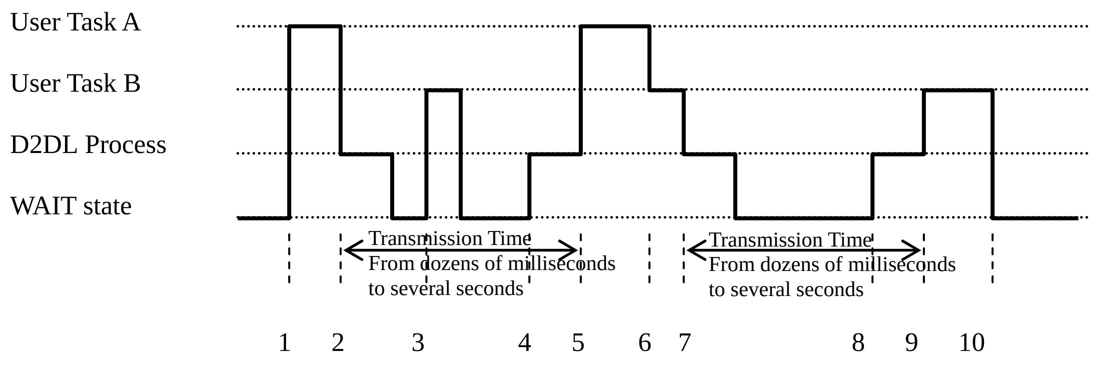
- By using the global variable:
  The callback function passes the transmission and reception result on to the user task which is waiting for the indication by using the global variable. The callback function sets the value to the global variable, and the user task refers to it.
  The following example uses the global variable.

```
[Global variable]
unsigned int rcv_stat = 0;        /* Reception result indication flag */

[User task process]
void UserTask2(void)
{
         :
     /* wait CB */
     while (rcv_stat == 0) {      /* Waiting for the indication
                                     of the callback function */
         tslp_tsk(1000);          /* Sleep for 1sec */
     }
     rcv_stat = 0;                /* Clear the indication flag */
     if (rcvBuffSize == 0)        /* Check the result
                                     of the callback function */
         :
}

[ Callback function process ]
void Receive_CB( d2dll_rcvParam *rcvParam )
{
         :
     rcvBuffSize = rcvParam->rcvDataLen;   /* Set the size
                                              of the received data */
     for ( i = 0; i < rcvBuffSize; i++ ) { /* Copy the received data */
        urbuf[i] = *(rcvParam->rcvData+i);
     }
         :
     rcv_stat = 1;                         /* Set the indication flag */
}
```

[Notes for using the callback function]
It takes dozens of milliseconds up to several seconds for the packet transmission and reception. If the callback process spends more than this time, the D2DL may fail the transmission and reception.
Additionally, if multiple tasks share the buffer etc., you should take care to overwrite the data and to change the status of variable.

7.2.1.5 Example of the action by the multiple user tasks

- Case: There are two tasks for the packet transmission(or reception)
  It simply makes a diagram and describes how to show the action when it wakes up the tasks (User Task A and B) together which call the transmission function which does not use the callback
  It assumes the User Task A wakes up first and the User Task B is next. The task priorities assume the same.

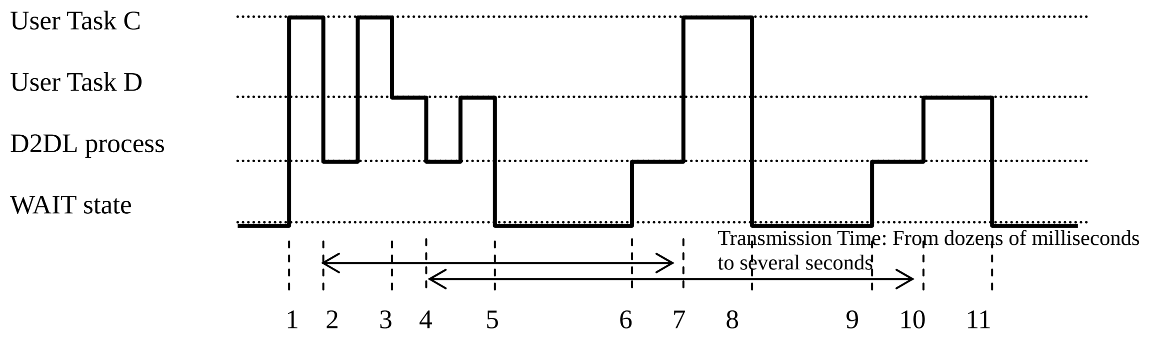


1. Wake up the User Task A. Call the transmission function, obtain one resource from the semaphore in the function, and then move to the D2DL process.
2. Start the transmission process of the D2DL. Then move to the WAIT state to get the transmission result.
3. Wake up the User Task B because the User Task A and the D2DL task moved to the WAIT state. It calls the transmission function, waiting for the semaphore in the function, and then moves to the WAIT state.
4. Determine the transmission result, and the D2DL wakes up the User Task A.
5. The User Task A gets the transmission result as the return value of the transmission function, its process has done, and exits. It has returned the resource to the semaphore which obtains at above process "1" at the end of the transmission function.
6. The transmission function of the User Task B obtains one resource from the semaphore, and moves to the D2DL process.
7. The transmission process of the D2DL starts, then it moves to the WAIT state to get the transmission result.
8. The transmission result is determined, and the User Task B is waked up by means of the D2DL.
9. The User Task B gets the transmission result by the return value of the transmission function, its process has done, and exits. It has returned the resource to the semaphore which obtains at above process "6" at the end of the transmission function.
10. The processes of the User Task A and the User Task B are finished.

Note: Do not release and exit the User Task WAIT state forcibly when it is triggered by the process steps "2", "3" and "7".

When it wakes up the tasks (the User Task A and B) together which calls the reception function (D2DLL_Recv) and does not use the callback, it is the same sequence. However the D2DLL_Recv function cannot assign the received packet to each destination application (the port number). Therefore it is not recommended that the multiple tasks call the reception function (D2DLL_Recv) in the multiple application environment.

- <u>Case: There are two tasks using the callback for the packet transmission (or reception)</u>
  It simply describes how to show the action the tasks (User Task C and D) which use the callback for packet transmission are waked up together.
  It assumes to implement different applications, the User Task C transmits via port 1 and the User Task D transmits via port 2. The task priorities assume the same.



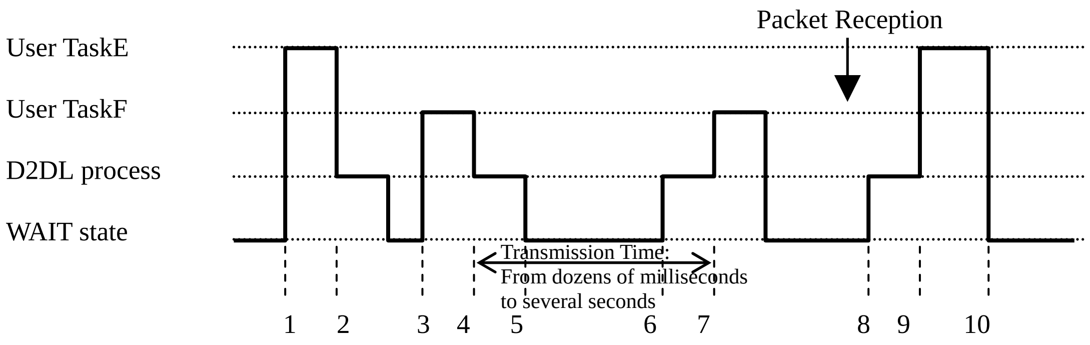
1. Wake up the User Task C. Call the transmission function, and then move to the D2DL process.
2. Start the transmission process of the D2DL (port 1). Return the function result to the User Task C because the transmission result will be passed by the callback function. The User Task C waits for the transmission result.
3. Wake up the User Task D because the User Task C and the D2DL task are in the WAIT state. Call the transmission function and move the D2DL process.
4. Start the transmission process of the D2DL (port 2). Return the function result to the User Task D because the transmission result will be passed by the callback function. The User Task D waits the transmission result.
5. Both of the User Task C and D return from the D2DL transmission function. Wait the transmission result for port 1 and port 2.
6. When the transmission result is determined form the port 1, the D2DL calls the callback function to indicate the User Task C.
7. The User Task C gets the transmission result by callback function, its process has done, and exits.
8. Wait the transmission result for the port 2.
9. When the transmission result is determined form the port 2, the D2DL calls the callback function to indicate the User Task D.
10. The User Task D gets the transmission result by callback function, its process has done, and exits.
11. The process of the User Task C and the User Task D have done

If it wakes up the tasks (User Task C and D) together which use the callback for reception, there is no action corresponding to the transmission function process from “1” to “5”. The timing of the User Task waking up and waiting and the action of the callback function are the same with the case of the transmission.

Every port can use the same transmission callback function registered for each port. In this case, you should identify the User Task to pass the transmission result in the callback function. The “sessionTag” is used for identification, for example the port number is set in the low 8 bits of the “sessionTag”.
It can be identified by the port ID “dstPortId” of the destination application in the received data for the reception callback function.

- <u>Case: there are the packet transmission task and the packet reception task</u>

  It simply describes to show the action when it wakes up the reception task (User Task E) and the transmission task (User Task F) together.
  It assumes that both of the User Task E and the User Task F do not use the callback function. The User Task E calls the reception API with the wait setting (D2DLL_INFINITE). The task priorities assume the same.



1. Wake up the User Task E. Call the reception function, obtain one resource from the semaphore for reception in the function, and then move to the D2DL process.
2. Start the reception process of the D2DL. Then move to the WAIT state to get the reception result.
3. Wake up the User Task F. It calls the transmission function, obtain one resource from the semaphore for transmission in the function, and then move to the D2DL process.

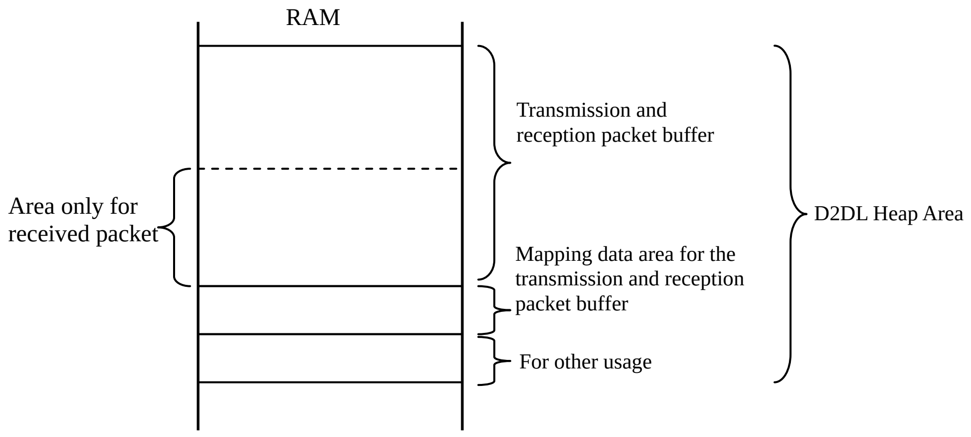
4. Start the transmission process of the D2DL. Then move to the WAIT state to get the transmission result.
5. Both of the User Task E and F are in the WAIT state in the D2DL function. They wait for the reception result or the transmission result.
6. The transmission result is determined, and the D2DL wakes up the User Task F.
7. The User Task F gets the transmission result as the return value of the transmission function, its process has done, and exits. It has returned the resource to the semaphore which obtains at above process “3” at the end of the transmission function.
8. The D2DL receives the data, creates the reception information, and wakes up the User Task E.
9. The User Task E gets the reception information by the argument of the reception function, its process has done, and exits. It has returned the resource to the semaphore which obtains at above process “1” at the end of the reception function.
10. The process of the User Task E and the User Task F have done

7.2.2 Addition of the Interrupt Handler

The interrupt process function is written in C. It is registered in the configuration file (CFG file) for using on the RTOS MR30 system. Please refer to the MR30 manual for the method of the interrupt process function definition and CFG file definition.
Note that it is not available to use the D2DL API in the interrupt process function.

7.2.3 Configuration of the D2DLHeap Area

The D2DL uses RAM area for the transmission and reception packet buffer, and other usages. (This area is called “the heap area” in this document.) The size of the heap area is configurable by the user and the allocation is as follows:


RAM
Transmission and reception packet buffer
Area only for received packet
D2DL Heap Area
Mapping data area for the transmission and reception packet buffer
For other usage

The packets which are requested to be transmitted by the user application and the packets which are received by PHY are temporarily stored in the ”transmission and reception packet buffer”. If the upper layer requests transmission continuously, for example, the transmission waiting packet occupies the buffer, so the user application cannot receive any packets. Therefore the “area only for received packet” is the area which is always allocated in the “transmission and reception packet buffer” for the received packet usage to avoid this condition.
“Mapping data area for the transmission and reception packet buffer” is the area which tracks the data location on the “transmission and reception packet buffer”.

Please set the size of these areas according to your system because they depend on the target system specification, such as the maximum payload size of the transmission and reception packet and the frequency of the transmission and reception.
The following calculated value gives an indication of the size which area used by one packet (transmission and reception packet) in the transmission and reception packet buffer.

“The payload size of the packet” + 17 (header size) + 101 (byte)
[Notes] You do not need to consider about the packet fragmentation on this calculation. For example, the 400 bytes payload packet occupies “400 + 17 + 101 = 518 bytes”.

The size of the data area for the transmission and reception packet buffer mapping is calculated as follows.
The transmission and reception packet buffer size (bytes) / 32

[Note] Round up the remainder.

"For other usage" area is fixed 284bytes.

You should allocate the total size of area which is added to each area in RAM, and set the size to the D2DL. The method of setting is as follows.

The heap area size is set by the following two methods.

- The parameter which is written in "crt0mr.a30"
- The parameters which are set by the function "D2DLL_SetParam"

Both of them should be set. First, the configuration by means of "crt0mr.a30" is described.

Please set the size of your heap area to the following gray shaded line in "crt0mr.a30" in hexadecimal. Do not change other item except this value in this file.

```
RAM_TOP_ADR           .EQU     400H
RAM_END_ADR           .EQU     63FFH         ; 24KB + 400H
ROM_TOP_ADR           .EQU     0E8000H       ; 96K ROM start

D2DLL_HEAP_AREA       .EQU 600H      ; 1.5K

;---------------------------------------------------------------
; Section allocation
;---------------------------------------------------------------

     :

;===============================================================
; Define the global variable (to be used by d2dll) here -- GB
;---------------------------------------------------------------
    _g_d2dll_StartOfHeap:       .BLKW   1
    _g_d2dll_EndOfHeap:         .BLKW   1

    .glb _g_d2dll_StartOfHeap
    .glb _g_d2dll_EndOfHeap

     :

;===============================================================
; initialize the global variables (to be used by d2dll) here - GB
;---------------------------------------------------------------
    mov.w   #__d2dll_heap_top, _g_d2dll_StartOfHeap
    mov.w   #__d2dll_heap_top + D2DLL_HEAP_AREA,_g_d2dll_EndOfHeap
    cmp.w   #RAM_END_ADR, _g_d2dll_EndOfHeap
    jgtu     _exit
```

The "_g_d2dll_StartOfHeap" is set to the top address of the heap area and the "_g_d2dll_EndOfHeap" is set to the end address of the heap area. These values are passed to the D2DL as the return value of the function "HALMemMngS_GetFreeRAMStart" and "HALMemMngS_GetFreeRAMEnd". Therefore you should implement these functions in the user application. (You can use the source code of the example program without any changes.)

[The function for getting the start point]
unsigned HALMemMngS_GetFreeRAMStart(void)

[The function for getting the end point]
unsigned HALMemMngS_GetFreeRAMEnd(void)

Next step describes the configuration by the function "D2DLL_SetParam".

Please set the size of your heap area by the following argument of the function “D2DLL_SetParam”.

The first argument	The second argument
D2DLL_IDX_DLL_MEMORY(71)	The size of the” transmission and reception packet buffer” (bytes)
D2DLL_IDX_RXRES_MEMORY (77)	The size of the “area only for received packet” (byte)

The example source code is as follows.

```
d2dllResult = D2DLL_SetParam( D2DLL_IDX_DLL_MEMORY, 916 );
if( d2dllResult != D2DLL_E_OK  ){
      while(1);
}
d2dllResult = D2DLL_SetParam( D2DLL_IDX_RXRES_MEMORY, 458 );
if( d2dllResult != D2DLL_E_OK  ){
      while(1);
}
```

[Note] These configurations should be set between calling the “D2DLL_Start” and the “D2DLL_Online” in the D2DL initialization sequence.

Please refer to the example program for more implementation details.

7.2.4 Modification of the MR30 Configuration File

When the user application uses the MR30 feature, you should modify the MR30 configuration file (CFG file). Please add the feature after referring under description because there are some items which are not available to change and have restriction to add.
The following gray shaded lines in Figure 1 are the configurations for the D2DL. Do not change the lines which do not have any comments.

[The description of the MR30 configuration file]

```
//************************************************************
//
// COPYRIGHT(C) 2005 RENESAS TECHNOLOGY CORPORATION
// AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
// OS Configuration File for M16C/6S DLL
//
//************************************************************

//----------------------------------------------
//  System Configuration
//----------------------------------------------
system{
    stack_size   = 330;
    priority     = 5;
    system_IPL   = 7;
    message_size = 16;
    timeout      = YES;
    task_pause   = NO;
};

clock{
    mpu_clock       = 15.36MHz;
    timer        = A2;
    IPL          = 6;
    unit_time       = 1ms;
    initial_time = 0;
};
```

<u>The definition about the system and the clock for the D2DL</u>
Do no changes except the system stack size "stack_size" and the maximum value of the task priority "priority".

```
//----------------------------------------------
//  Task Configuration
//----------------------------------------------
task[]{
    entry_address    = Main_WorkerThread();
    stack_size       = 280;
    priority     = 4;
    initial_start    = ON;
};
task[]{
    entry_address    = D2dll_Work();
    stack_size       = 160;
    priority     = 3;
    initial_start    = OFF;
};
task[]{
    entry_address    = D2dll_Recv();
    stack_size       = 180;
    priority     = 1;
    initial_start    = OFF;
};
task[]{
    entry_address    = D2dll_Send();
    stack_size       = 180;
```

<u>The definition of the task for the D2DL</u>
Do not change the configuration and the order.
Main_WorkerThread() is the user task(Main Task), so it is available to change the stack size.
And maximum stack size of callback functions should be added to the stack size of D2dll_Work (160) because the callback functions are called by D2dll_Work Task.

```
    priority     = 2;
    initial_start    = OFF;
};

task[]{
    entry_address    = Stsk_port1();
    stack_size       = 180;
    priority     = 5;
    initial_start    = OFF;
};
task[]{
    entry_address    = Stsk_port2();
    stack_size       = 180;
    priority     = 5;
    initial_start    = OFF;
};
task[]{
    entry_address    = RecvTask();
    stack_size       = 180;
    priority     = 5;
    initial_start    = OFF;
};

//--------------------------------------------
//  Flag Configuration
//--------------------------------------------
flag[]{
    name     = _f1;
};
flag[]{
    name     = _f2;
};
flag[]{
    name     = _f3;
};
flag[]{
    name     = _f4;
};
flag[]{
    name     = _f5;
};
flag[]{
    name     = _f6;
};
flag[]{
    name     = _f7;
};
flag[]{              // UART0 Receive
    name     = Main_pEventIpcRequest;
};
flag[] {
    name     = UserTaskState;
};

//--------------------------------------------
//  Semaphore Configuration
//--------------------------------------------
semaphore[]{
    name     = _s1;
    initial_count    = 1;
```

<u>The definition for the user application task</u>
If you add the task, please make sure to define it after the D2DL task definition.

<u>The definition of the event flag for the D2DL</u>
Do not change the configuration and the order.

<u>The definition of the event flag for the user application</u>
Please make sure to define after the D2DL event flag definition.

```
};
semaphore[]{
   name    = _s2;
   initial_count    = 1;
};
semaphore[]{
   name    = _s3;
   initial_count    = 1;
};
semaphore[]{
   name    = _s4;
   initial_count    = 1;
};
semaphore[]{
   name    = _s5;
   initial_count    = 1;
};
semaphore[]{
   name    = _s6;
   initial_count    = 1;
};
semaphore[]{
   name    = _s7;
   initial_count    = 1;
};
semaphore[]{
   name    = _s8;
   initial_count    = 1;
};
semaphore[]{
   name    = _s9;
   initial_count    = 1;
};
semaphore[]{
   name    = _s10;
   initial_count    = 1;
};
semaphore[]{
   name    = _s11;
   initial_count    = 1;
};
semaphore[]{
   name    = _s12;
   initial_count    = 1;
};
semaphore[]{
   name    = _s13;
   initial_count    = 1;
};
semaphore[]{
   name    = _s14;
   initial_count    = 1;
};
semaphore[]{            // user
   name    = uartTx_lock;
   initial_count    = 1;
};
```

<ins>The definition of the semaphore for the D2DL</ins>
Do not change the configuration and the order.

<ins>The definition of the semaphore for the user application</ins>
Please make sure to define after the D2DL semaphore definition.

```
//-------------------------------------------
//  Interrupt Handler
//-------------------------------------------
interrupt_vector[8]{                // (SI/O4)
   entry_address       = D2dll_PhyRx_PhyUsartISR();
   os_int           = YES;
};
interrupt_vector[18]{           // UART0 Rx  // user
   entry_address       = _DebugPort_ReadByte_a30;
   os_int           = YES;
};
interrupt_vector[21]{           // (TA0)
   entry_address       = D2dll_Timer_ISR();
   os_int           = YES;
};
interrupt_vector[22]{           // (TA1)
   entry_address       = D2dll_Timer1_ISR();
   os_int           = YES;
};
interrupt_vector[29]{           // (INT0)
   entry_address       = D2dll_Phy_ISR();
   os_int           = YES;
};

interrupt_vector[10]{           // (BCN)
   entry_address       = sis_int();
   os_int           = YES;
};
interrupt_vector[15]{           // (S2T)
   entry_address       = sit_int();
   os_int           = YES;
};
interrupt_vector[16]{           // (S2R)
   entry_address       = sir_int();
   os_int           = YES;
};

//
// End of Configuration
//
```

The definition of the interrupt vector for the D2DL
Do not change the configuration.

Figure 1 : The example of the MR30 configuration file

- System Timer
  It isn't available to change except the system stack size "stack_size" and the maximum priority of the task "priority".
  Please calculate the system stack size with the stack size calculation program "StkViewer", for example. If you don't add any interrupt handlers, the size is 330.
  Please set the maximum value of the priority for the user task to the maximum value of the task priority".
  If you don't add the user task, the value is 4.

- System Clock
  All item cannot be changed.

- Task
  The above gray shaded lines of the task[] {…} in Figure 1 are the definition of the task used in the D2DL and the main task (Main_WorkerThread()) of the user application. Do not change the line except the stack size of the user main task and D2dll_Work task and the priority of the user main task. Do not change the order of the configuration.
  The user main task "Main_WorkerThread()" is a task which is waked up at first after power on. Please

define the D2DL initialization, the user application initialization and wake-up the other task, etc. Calculate the stack size "stack_size" with the stack size calculation program "StkViewer", for example, and set the task priority "priority" the value 4 and above.
When you add the task used by the user application, add it after the definition of the task using in the D2DL and the user main task. The method of configuration of the stack size and the priority are the same as the method for the user main task.
Do not set the ID number in "[]" of the task[].

- Event flag
  The gray shaded lines of the flag[] {…} in Figure 1 are the definition of the event flag used in the D2DL. Do not change all lines. When the user application uses the event flag feature, please add it after the definition of the event flag used in the D2DL. Do not set the ID number in "[]" of the flag[].

- Semaphore
  The above gray shaded lines of the semaphore [] {…} in Figure 1 are the definition of the semaphore used in the D2DL. Do not change all lines.
  When the user application uses the semaphore feature, please add it after the definition of the semaphore used in the D2DL. Do not set the ID number in "[]" of the semaphore [].

- Interrupt Vector
  The above gray shaded lines of the interrupt_vector [ID number] {…} in Figure 1 are the definition of the interrupt vector used in the D2DL. Do not change all lines.
  Please define the interrupt vector used in the user application with setting the vector number. If it is the Watchdog Timer interrupt, you should set "NO" to the OS-dependent interrupt handler "os_int", and if it is the other interrupt, you should set "YES". Please note that it is not available to call the D2DL API in the interrupt handler.

The configurator "cfg30" makes "id.h" with the configuration file. (The cfg30 is executed in the HEW.) Please verify the generated "id.h" with the following "define" definition when you add the tasks, event flags or semaphore for the user application.

[id.h description]

```
#define ID_Main_WorkerThread 1
#define ID_D2dll_Work    2
#define ID_D2dll_Recv    3
#define ID_D2dll_Send    4
<<Hereafter, your task IDs are defined. >>
    :
#define ID__f1      1
#define ID__f2      2
#define ID__f3      3
#define ID__f4      4
#define ID__f5      5
#define ID__f6      6
#define ID__f7      7
<<Hereafter, your flag IDs are defined. >>
    :
#define ID__s1      1
#define ID__s2      2
#define ID__s3      3
#define ID__s4      4
#define ID__s5      5
#define ID__s6      6
#define ID__s7      7
#define ID__s8      8
#define ID__s9      9
#define ID__s10     10
#define ID__s11     11
#define ID__s12     12
#define ID__s13     13
```

```
#define ID__s14      14
<<Hereafter, your semaphore IDs are defined. >>
   :
```

7.3 Notes

7.3.1 Influence of the D2DL task and its interrupt process on the user application

During the main function "Main_WorkerThread" of the user application or the new interrupt functions which are added by the user is running, **the interrupt process for the D2DL** may start up and then **the tasks in the D2DL library** run circumstantially.
In this case, **the user application pauses** and **the process of the D2DL library starts to run**. The occurrence frequency of this D2DL interrupt process (which includes the D2DL task process) depends on the amount of the packets on the PLC.

7.3.2 Notes on the interrupt

- If the processing time of the interrupt process of the user application meets one of the following conditions, we recommend enabling the multiple interrupt which priority level is 3 or more. If you do not enable it, you may miss some packets.

  <u>Condition: the processing time per one interrupt process of the user application is 100 us or more, and its interrupt may occurs 2.4 ms interval or less</u>

  <u>Condition 2 : the processing time of one interrupt process of the user application is longer than 500 us</u>

- The interrupt processing time of the D2DL and its occurrence timing are as follows.

Trigger of the interrupt	Function name	Processing time	Occurrence timing
SI/O4	D2dll_PhyRx_PhyUsartISR	20 us – 500 us	The transmission and reception on the PLC. (In other case, it may occur at random times by the reception of the noise.)
INT0	D2dll_Phy_ISR	20 us – 120 us	
TA0	D2dll_Timer_ISR	30 us – 90 us	
TA1	D2dll_Timer1_ISR	10 us or less	

[Notes] The SI/O4 interrupt process takes about 500 us only in case of the reception for the top byte of the packet. In other case, the process takes about 20 us.

- To disable the interrupt in the user application, please select one of the following methods:
  I. Disable the interrupt by the operation of the interrupt enable flag (I flag) and the processor interrupt priority level (IPL).
  Please make sure to use the interrupt disabling and enabling in pairs.
  II. Change the interrupt priority level (from ILVL2 to 0) to disable
  Please clear the I flag before and after the operation of the interrupt priority level (from ILVL2 to 0).

- The D2DL reserves the interrupt number from 32 to 47 of the INT instruction. Please use the interrupt number except from 32 to 47 when you use the software interrupt in the user application.

- Register Bank Usage
  It is not recommended to change the register bank.

7.3.3 Notes on the debug

- You can set up to two breakpoints at the same time (in case of using the KD30).
- Do not use the DBC interrupt because it is the specific interrupt for the developer support tool.

7.3.4 Notes on the multi task

- The D2DL APIs can be called only in the task. They cannot be called in the interrupt handler. If you call the API in the interrupt handler, it becomes the indefinite semantics.
- When you make the application program with this library, you should set the user task priority which is used by the application to value 4 or above, and lower than the internal tasks of the D2DL.
- Do not wake up the user task, while the D2DL does it in sleep state, by the user application. If you wake it up, it causes illegal semantics.

- You have to use the RENESAS RTOS MR30 for using the D2DL multi task edition library. Please install the MR30 in the same directory with the NC30WA. When you upgrade the NC30, you should install it in the same directory with the previous version and you should not install it in the default directory of the NC30 installer. If the install directory is different between NC30WA and MR30, it will cause build errors.

8. Description of the example program

This section describes the outline of the example program.

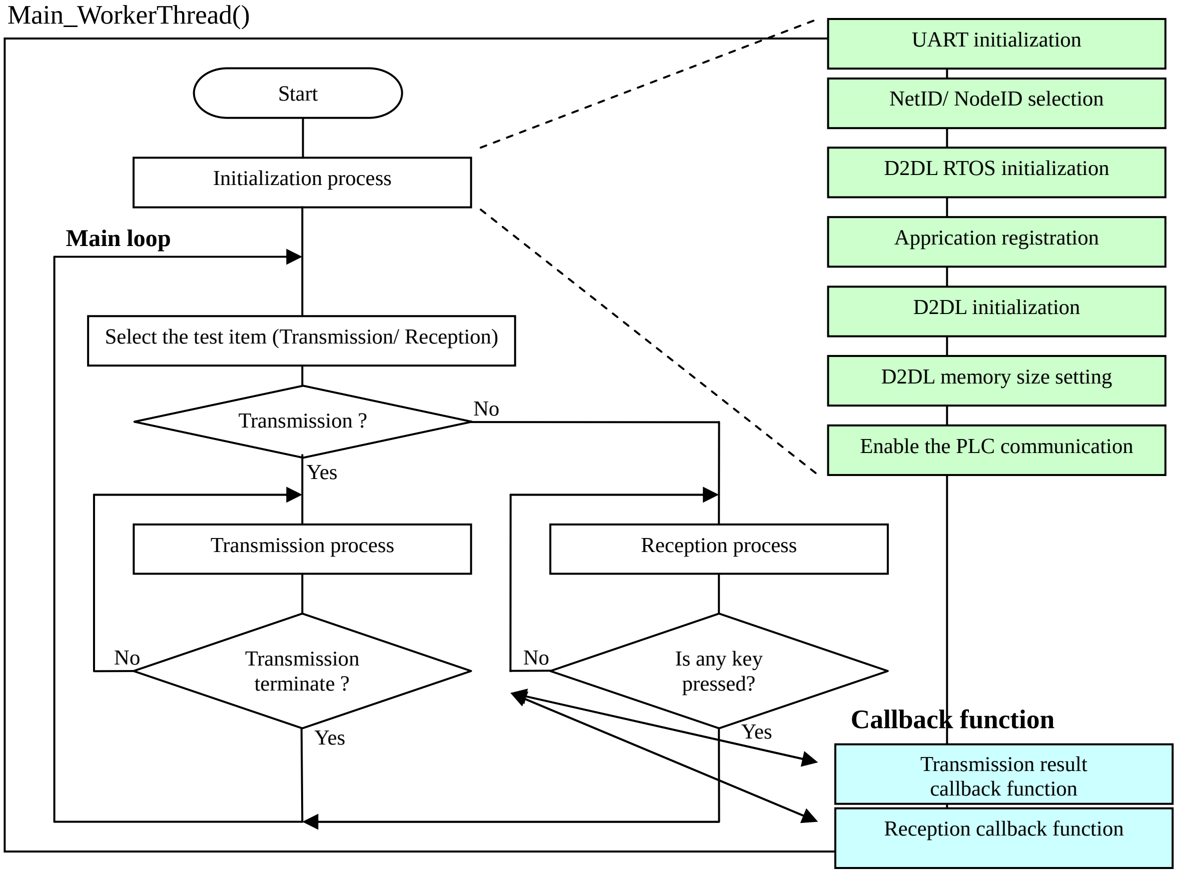
8.1 Overview

- It is a simple program for communication test.
- Connect each node of the M16C/6S target board to your PC via RS-232C cable (crossover cable), and launch a terminal software (for example, HyperTerminal).
- The following menu is shown on the terminal window. If you select the mode from 1 to 5, you can confirm the communication status between each node.

```
------- D2DLL Test Program (V1.00) -------
Copyright Renesas Technology Corporation and
                              Renesas Solutions Corporation

 << note >>   Use Number key & Back space.

Enter Network ID (1-1023) > 7
Enter Source Node ID (1-2047) > 2
Enter Destination Node ID (1-2047) > 5

D2DLL Initialize.....     Successful.

   ******** PLC TEST MENU ********
   1. Transmit 100packets by UniCast (40byte)
   2. Transmit 100packets by BroadCast (40byte)
   3. Transmit 500packets by UniCast (40byte)
   4. Transmit 500packets by BroadCast (40byte)
   5. Count Received Packets
Select Mode >
```

8.2 General flowchart

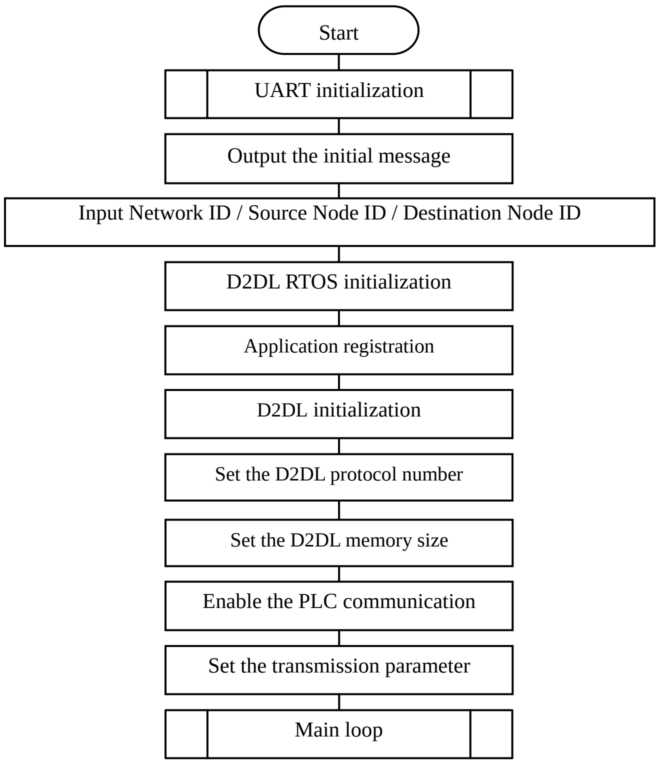
Main_WorkerThread()



8.3 Details of the program

8.3.1 Main function - Main_WorkerThread()

8.3.1.1 Flowchart


Start
UART initialization
Output the initial message
Input Network ID / Source Node ID / Destination Node ID
D2DL RTOS initialization
Application registration
D2DL initialization
Set the D2DL protocol number
Set the D2DL memory size
Enable the PLC communication
Set the transmission parameter
Main loop

8.3.1.2 Details of the function

```
void Main_WorkerThread ( void )
{
         :
         :
     //----------------------------------------------
     // Initialize Uart&LED
     //----------------------------------------------
     _Uart_Init();                                                    ---  1

     //----------------------------------------------
     // Select NetID/NodeID
     //----------------------------------------------
     printf("¥n¥n");                                                  ⎫
     printf("------- D2DLL Test Program (V1.00) -------¥n");          ⎪
     printf("Copyright Renesas Technology Corporation and¥n");        ⎬ ---  2
     printf("                   Renesas Solutions Corporation¥n¥n");  ⎪
     printf(" << note >>  Use Number key & Back space.¥n¥n");         ⎭

     // Input NetID/NodeID
     printf("Enter Network ID (1-1023) > ");       // NetID
     UartStrGet(str);                                                 ---  3
     src_LNetId = str2DN(str);
     printf("¥n");
     printf("Enter Source Node ID (1-2047) > ");   // Src NodeID
     UartStrGet(str);                                                 ---  4
     src_NodeId = str2DN(str);
     printf("¥n");
     printf("Enter Destination Node ID (1-2047) > ");// Dst NodeID
     UartStrGet(str);                                                 ---  5
     dst_NodeId = str2DN(str);
     printf("¥n¥n");
     printf("D2DLL Initialize.....   ");
```

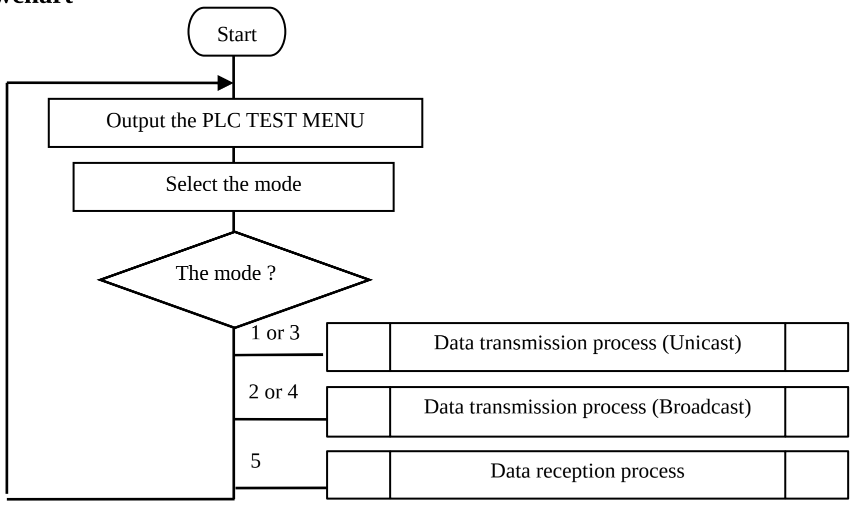
```
//-------------------------------------------
//  D2DLL initialize
//-------------------------------------------
d2dllResult = D2DLL_Init( NULL );                                    ---  6
if( d2dllResult != D2DLL_E_OK ){
   while( TRUE ){
      printf("Failed.¥n¥n");
   }
}
//-------------------------------------------
// Application Register
//-------------------------------------------
d2dllResult = D2DLL_RegApp( 1, User_Receive_CB, User_SendResult_CB ); ---  7
if( d2dllResult != D2DLL_E_OK ){
   while( TRUE ){
      printf("Failed.¥n¥n");
   }
}
//-------------------------------------------
// D2DLL Start
//-------------------------------------------
dsn[0] = (uchar)src_NodeId;
dsn[1] = (uchar)(src_NodeId >> 8);
d2dllResult = D2DLL_Start( src_LNetId, src_NodeId, D2DLL_RXTP_MINE,
                           D2DLL_REGION_JPN, dsn, D2DLL_DIS_REP);    ---  8
if( d2dllResult != D2DLL_E_OK  ){
   while( TRUE ){
      printf("Failed.¥n¥n");
   }
}
//-------------------------------------------
// Set the Protocol Version
//-------------------------------------------
d2dllResult = D2DLL_SetParam(D2DLL_IDX_PROTOCOL_VER, 0x3B );         ---  9
if( d2dllResult != D2DLL_E_OK  ){
   while( TRUE ){
      printf("Failed.¥n¥n");
   }
}
//-------------------------------------------
// Set the size of DLL Memory
//-------------------------------------------
d2dllResult = D2DLL_SetParam( D2DLL_IDX_DLL_MEMORY, 916 );           ---  10
if( d2dllResult != D2DLL_E_OK  ){
   while( TRUE ){
      printf("Failed.¥n¥n");
   }
}
d2dllResult = D2DLL_SetParam( D2DLL_IDX_RXRES_MEMORY, 458 );         ---  11
if( d2dllResult != D2DLL_E_OK  ){
   while( TRUE ){
      printf("Failed.¥n¥n");
   }
}
//-------------------------------------------
// Online
//-------------------------------------------
d2dllResult = D2DLL_Online();                                        ---  12
if( d2dllResult != D2DLL_E_OK  ){
   while( TRUE ){
      printf("Failed.¥n¥n");
   }
}
//-------------------------------------------
// Send Option Set
//-------------------------------------------
d2dllResult = D2DLL_SendOption( 2, 0, D2DLL_SND_AUTO );              ---  13
```

```
     if( d2dllResult != D2DLL_E_OK  ){
        while( TRUE ){
           printf("Failed.¥n¥n");
        }
     }
     printf("Successful.¥n");
     //-------------------------------------------
     // The main processing loop
     //-------------------------------------------
     UserApplication();                                               ---  14
}
```

1. Initialize the UART.
2. Output the initial menu.
3. Get the network ID via the user input.
4. Get the source node ID via the user input.
5. Get the destination node ID via the user input.
6. Initialize the D2DL RTOS by the D2DLL_Init API function.
7. Register the application by the D2DLL_RegApp API function. It registers the User_Receive_CB function as a reception callback function, and User_SendResult_CB function as a transmission callback function. (It describes later about each callback function.)
8. Initialize the D2DL by the D2DLL_Start API function.
9. Set the dedicated protocol number by the D2DLL_SetParam API function (if you do not use the echonet).
10. Set the size of the sent/received buffer of the D2DL by the D2DLL_SetParam API function.
11. Set the size of the special area for the received packet of the D2DL.
12. Enable the PLC communication by the D2DLL_Online API function.
13. Set the transmission parameter by the D2DLL_SendOption API function.
14. Call the UserApplication function for the main loop process. (It describes later.)

8.3.2 UserApplication function

8.3.2.1 Flowchart



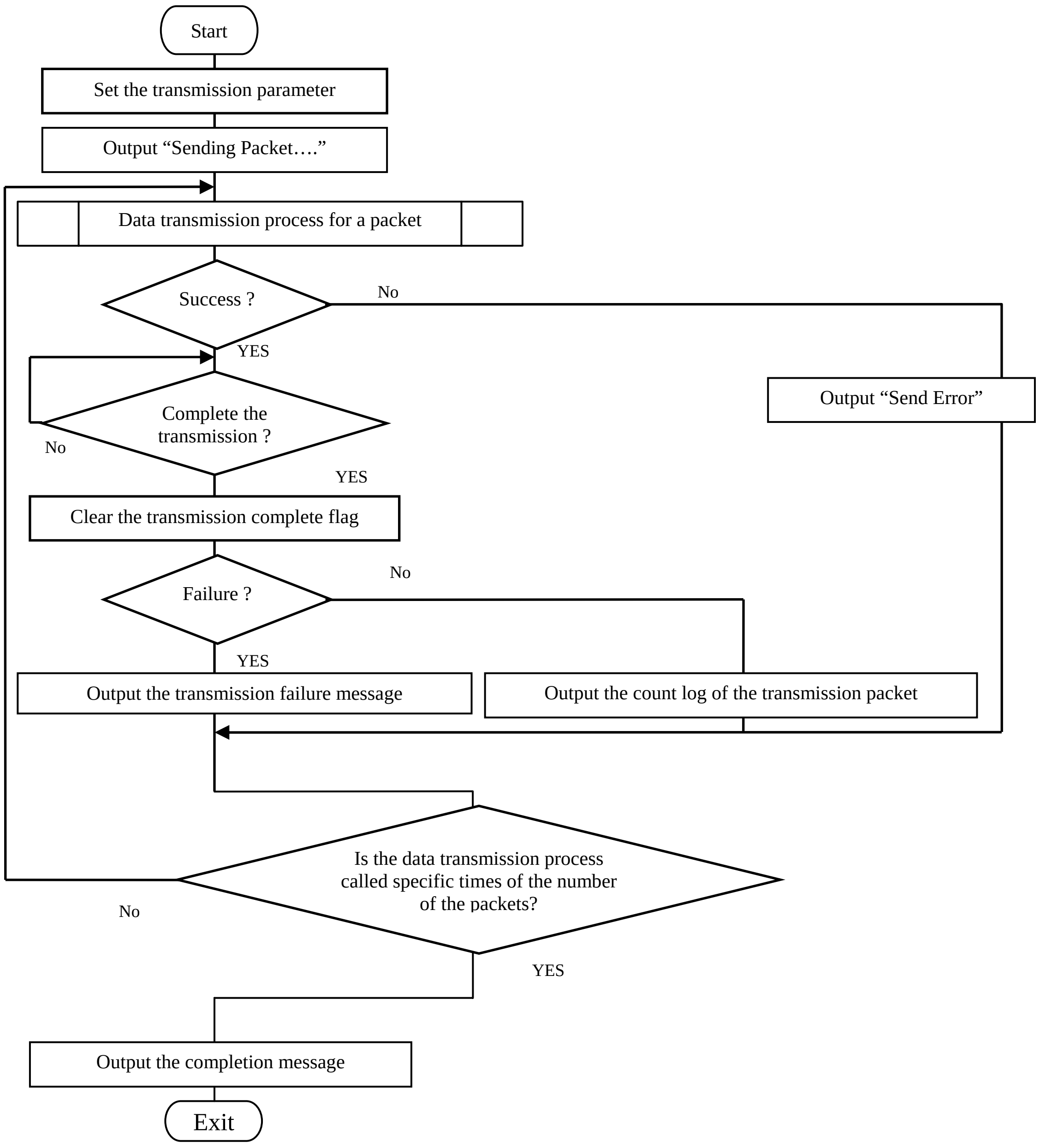
8.3.2.2 Details of the function

```
static bool_v UserApplication(void)
{
    uchar    str[17];
    uint16   command;

    while ( TRUE ) {
        printf("¥n");
        printf("  ******** PLC TEST MENU ********¥n");                    ⎫
        printf("  1. Transmit 100packets by UniCast (40byte)¥n");          ⎪
        printf("  2. Transmit 100packets by BroadCast (40byte)¥n");        ⎪
        printf("  3. Transmit 500packets by UniCast (40byte)¥n");          ⎬  --- 1
        printf("  4. Transmit 500packets by BroadCast (40byte)¥n");        ⎪
        printf("  5. Count Received Packets¥n");                           ⎪
        printf("Select Mode > ");                                          ⎭
        UartStrGet(str);                                                      --- 2
        printf("¥n");
        command = str2DN(str);
        switch (command) {                                                    --- 3
            case 1:
            case 3:
                It_User_Send(dst_NodeId, command - 1);                        --- 4
                break;
            case 2:
            case 4:
                It_User_Send(0, command - 1);                                 --- 5
                break;
            case 5:
                It_User_recept();                                             --- 6
                break;
            default:
                break;
        }
    }
}
```

1. Output the PLC TEST MENU.
2. Get the user selected mode number.
3. Branch each process by the mode number.
4. Call the It_User_Send function for the unicast transmission. (It is described later.)
5. Call the It_User_Send function for the broadcast function. (It is described later.)
6. Call the It_User_recept function for the reception. (It is described later.)

8.3.3 It_User_Send function

8.3.3.1 Flowchart


Start
Set the transmission parameter
Output "Sending Packet…."
Data transmission process for a packet
Success ?
No
YES
Output "Send Error"
Complete the transmission ?
No
YES
Clear the transmission complete flag
Failure ?
No
YES
Output the transmission failure message
Output the count log of the transmission packet
Is the data transmission process called specific times of the number of the packets?
No
YES
Output the completion message
Exit

8.3.3.2 Details of the function

```
static void It_User_Send(uint16 nodeId, uint16 parm)
{
    uint16 i;
    sint16 errStatus=0;
    d2dll_sndParam sndParam;

    sndParam.srcPortId = 1;                          ⎫
    sndParam.dstNodeId = nodeId;                     ⎪
    sndParam.dstPortId = 1;                          ⎪
    sndParam.sndData = TxData;                       ⎪
    sndParam.sndDataLen = tx_param[parm].lData;      ⎬        ---  1
    sndParam.sessionTag = 1;                         ⎪
    sndParam.sndPrty = D2DLL_PRTY_LOW;               ⎪
    sndParam.sndAck = D2DLL_SND_ACKD;                ⎭
```

```
     printf("Sending Packet....  ");
     for (i = 0; i < tx_param[parm].sndCnt; i++) {           ---  2
        /* send */
        errStatus = D2DLL_Send( &sndParam );                 ---  3
        if( errStatus == D2DLL_E_OK ){                       ---  4
                  while( bIs_SendFinished != TRUE );         ---  5
                  bIs_SendFinished = FALSE;                  ---  6

                  if (sendStatus != D2DLL_E_OK ) {
                     printf("Failed(%d).¥n", sendStatus);   ---  7
                  }
                  else {
                     printf("%d", i % 10);                   ---  8
                  }
        }
        else{
           printf("Send Error.¥n");                          ---  9
        }
     }
     printf(" Complete!¥n");                                 ---  10
}
```

1. Set the transmission parameter.
2. Loop iteration which is the specific times of the number of the packets.
3. Send a packet with the D2DLL_Send API function.
4. Check if the D2DLL_Send API function exits successfully or not.
5. Loop iteration until it has transmitted. The transmission complete flag “bIs_SendFinished” is set to TRUE in the callback function because the User_SendResult_CB function is called when the D2DL finishes the transmission.
6. The transmission complete flag “bIs_SendFinished” is set to FALSE to clear it.
7. Output the error message if the transmission result is not normal.
8. Output the count log of the transmission packet if the transmission result is not normal.
9. Output the error message if the D2DLL_Send API function does not terminate normally.
10. Output the completion message after loop iteration which is the specific times of the number of the packets.

8.3.4 User_SendResult_CB function

8.3.4.1 Details of the function

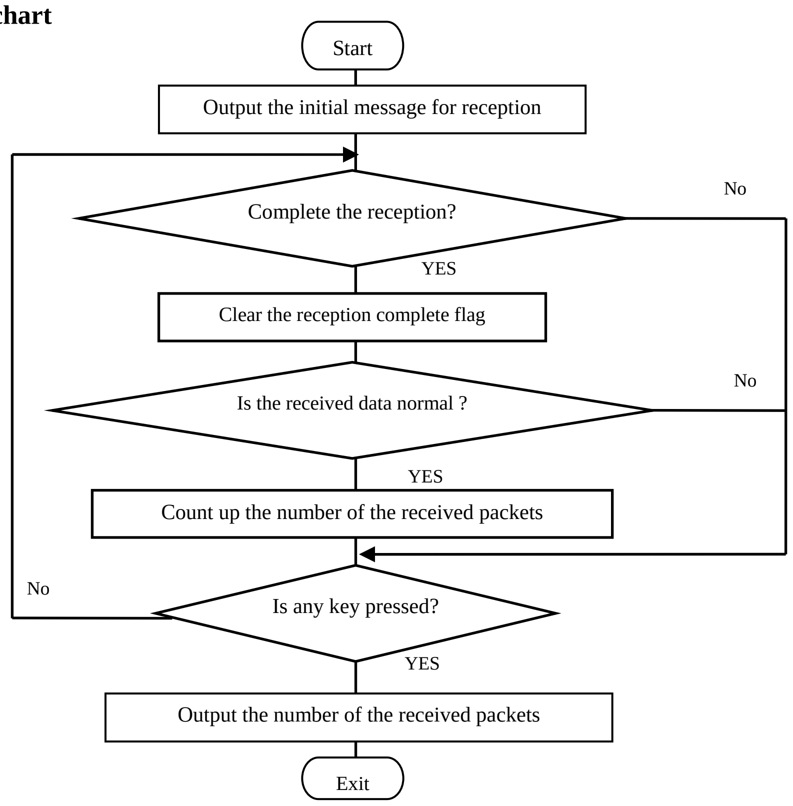
This is a callback function. It is called when the D2DL finishes the transmission.

```
static void User_SendResult_CB( uint16 sessionTag, sint16 sndResult )
{
   sendStatus = sndResult;                                      ---  1
   bIs_SendFinished = TRUE;                                     ---  2
}
```

1. Store the transmission result to “sendStatus”
2. Set the transmission complete flag “bIs_SendFinished” to TRUE

8.3.5 It_User_recept function

8.3.5.1 Flowchart


Start
Output the initial message for reception
Complete the reception?
No
YES
Clear the reception complete flag
Is the received data normal ?
No
YES
Count up the number of the received packets
No
Is any key pressed?
YES
Output the number of the received packets
Exit

8.3.5.2 Details of the function

```
static void It_User_recept( void )
{
     RxParam rx_param;
     uint16 pren = n_uart_buff;
     uchar str[8];

     printf("Counting Received Packet.... ¥n ");          }
     printf("Enter any key to stop counting¥n¥n");        }    ---  1
     rx_param.m_total_cnt = 0;                            }

     while (pren == n_uart_buff) {                             ---  2
        if( bIs_ReceiveFinished == TRUE ){                     ---  3
          bIs_ReceiveFinished = FALSE;                         ---  4
          if (0 == CmpRcvDat(TxData, urbuf, rcvBuffSize)) {    ---  5
             if ( rcvBuffSize == SNDLEN ) {
                 rx_param.m_total_cnt++;                       ---  6
             }
          }
        }
     }
     Main_pEventIpcRequest = FALSE;
     n_uart_buff = 0;

     printf("¥n");                                        }
     printf("%d Packets Received¥n", rx_param.m_total_cnt); }  ---  7
     printf("¥n");                                        }
}
```

1. Output the initial message for reception.
2. Wait until any key is pressed.
3. Check the reception complete flag “bIs_ReceiveFinished”. The reception complete flag “bIs_ReceiveFinished” is set to TRUE in the callback function because the User_Receive_CB function is

called when the D2DL finishes the reception.
4. The reception complete flag “bIs_ReceiveFinished” is set to FALSE to clear it.
5. Verify the received data with the transmission data of this example program.
6. Count up the number of received packets.
7. Output the number of received packets.

8.3.6 User_Receive_CB function

8.3.6.1 Details of the function

This is a reception callback function. It is called when the D2DL finishes the reception.

```
static void User_Receive_CB( d2dll_rcvParam *rcvParam )
{
   sint16 i;

   rcvBuffSize = rcvParam->rcvDataLen;                  ---  1
   //data transfer
   for ( i = 0; i < rcvBuffSize; i++ ) {                ---  2
      urbuf[i] = *(rcvParam->rcvData+i);
   }
   bIs_ReceiveFinished = TRUE;                          ---  3
}
```

1. Store the size of the received packet in rcvBuffSize.
2. Copy the received packet to the buffer.
3. The reception complete flag “bIs_ReceiveFinished” is set to TRUE.

8.3.7 D2DLL_CB_RxLEDon function

8.3.7.1 Details of the function

This function makes the LED on.

```
void D2DLL_CB_RxLEDon(void)
{
   PD8_1 = 1;                                           ---  1
   P8_1 = 1;                                            ---  2
}
```

1. Set the port 81 direction register output.
2. Set the port 81 on.

8.3.8 D2DLL_CB_RxLEDoff function

8.3.8.1 Details of the function

This function makes the LED off.

```
void D2DLL_CB_RxLEDoff(void)
{
   P8_1 = 0;                                            ---  1
}
```

1. Set the port 81 off.

9. Development environment

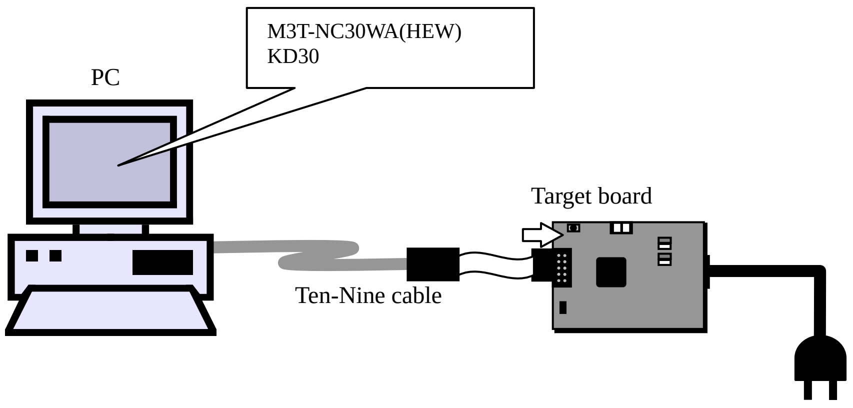
9.1 Development for using the simple debugger KD30

The following shows a list of essential item to develop the user application using the simple debugger KD30.

Item Name	Description	Recommended Ver. No.
C Compiler Package M3T-NC30WA	This package includes the C compiler, assembler and linker.	Ver 5.30 Release 02 or Ver 5.40 Release 00
High-performance Embedded Workshop (HEW)	The D2DL example program is provided as HEW project environment. The development based on this program is recommended. The HEW is included in the NC30WA package.	Ver 4
Simple Debugger KD30	It communicates with the monitor program on the chip, so provides the debug environment on PC. You can download it from the RENESAS web site for free.	V 4.10
Monitor program for M16C/6S	This program is written in the chip for debug by KD30.	V 1.0
Flash Programmer M3A-0806	The included cable (Ten-Nine cable) is used for connecting for debug.	--
D2DL Library	This product.	--
M16C/6S module	This is the target board.	--

[Note] You can also use the M3A-0665 FoUSB for writing to the flash memory.

The development environment using the KD30 is shown as follows.



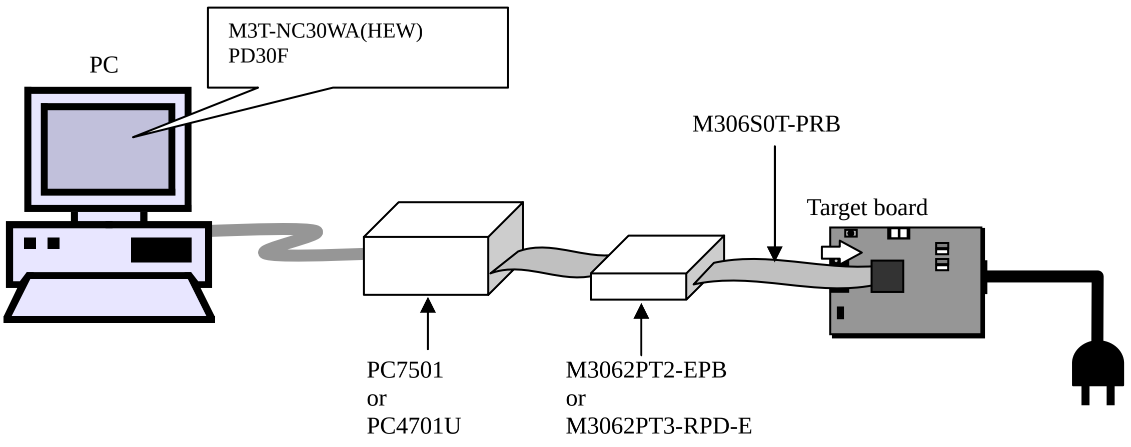
9.2 Development for using the emulator debugger PD30F

The following shows a list of essential item to develop the user application using the simple debugger PD30.

Item Name		Description	Recommended Ver. No.
C Compiler Package M3T-NC30WA		This package includes the C compiler, assembler and linker.	Ver 5.30 Release 02 or Ver 5.40 Release 00
High-performance Embedded Workshop (HEW)		The D2DL example program is provided as HEW project environment. The development based on this program is recommended. The HEW is included in the NC30WA package.	Ver 4
1	Emulator Debugger PC7501	These are the emulator system. Please use a set either 1 or 2.	--
	Emulation Probe M3062PT2-EPB		--
2	Emulator Debugger PC4701U		--
	Emulation Pod M3062PT3-RPD-E		--
Emulator Debugger PD30F		This is attached to the emulator.	--
Signal Converter Board for M16C/6S Group M306S0T-PRB		This is connected to the tip of the pod or the probe. It converts the signal for M16C/6S.	--
Flash Programmer M3A-0806		It is used for writing to the flash memory.	--
D2DL Library		This product.	--
M16C/6S module		This is the target board. Notes: **A socket should be set on the M16C/6S.** Please refer the M306S0T-PRB User's Manual for details. (It can be downloaded from the RENESAS web site.)	--

[Note] You can also use the M3A-0665 FoUSB for writing to the flash memory.

The development environment using the Emulator Debugger is shown as follows.



M16C/6S DATA LINK LAYER LIBRARY D2DL USER'S MANUAL

Publication Date:	Jun. 30, 2006	Rev.1.02
Published by:	Sales Strategic Planning Div. Renesas Technology Corp.	
Edited by:	Application Engineering Department 3 Renesas Solutions Corp.	

M16C/6S DATA LINK LAYER LIBRARY
D2DL USER'S MANUAL

Renesas Technology Corp.
2-6-2, Ote-machi, Chiyoda-ku, Tokyo, 100-0004, Japan